U.S. Patent Application

for

METHOD FOR FACILITATING PRICING, SALE AND DISTRIBUTION OF FUEL TO A CUSTOMER

INVENTORS:

Louis MORRISON, III
Scott WILLIAMS
Susan SIMS
Jamie DULANEY
Leslie VANDAGRIFF
Gail WOODY
Shelly HARLAN
Al SHIELDS
Rick SURLES
Neil STRONG
Darin MORSE

Attorney Docket No.

068.0003

TITLE OF THE INVENTION

METHOD FOR FACILITATING PRICING, SALE AND DISTRIBUTION OF FUEL TO A CUSTOMER

COMPUTER SOFTWARE ADDENDUM

Attached hereto is a compact disc containing computer software and data including executable programs, scripts, and database management system tables that are used to implement the systems and methods provided by the present invention. More particularly, the attached compact disc contains software and data used to implement at least two distinct applications comprising the systems and methods provided by the present invention; such two distinct applications include a broad-based, general use energy management system (referred to as the Energy Management System "EMS"), and a limited user/function restricted application (referred to as the Producer Control Center "PCC") intended for use by fuel producers needing access to centrally stored and managed fuel deal data. Such material is protected by the Copyright Laws of the United States (17 U.S.C. § 101, *et seq.*) and may not be copied without the express, written authorization of the copyright holder (Highland Energy Corporation). Copyright © 2001, Highland Energy Corporation. All Rights Reserved.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to systems and methods used to facilitate pricing, sale, and distribution of fuel to a customer. More particularly, the present invention is directed to automated systems and methods that are used to price fuels such as natural gas, oil, gas, other petroleum based fuels, etc., to facilitate commodity sales of such fuels, and to distribute such fuels to customers, and to track and report sales and distribution related data.

Description of the Related Art

Fuel sales and distribution systems and techniques are well known. Everyday millions of fuel sale contracts are completed in the U.S. and abroad. Fuels produced by a range of producers are transported over many modes of transportation (e.g., gas pipelines, etc.) to ultimately arrive at an intended destination. The steps involved in pricing fuel, selling a reserve, storing reserves, and ultimately transporting purchased fuel involve many parties including producers, agents, brokers, other middlemen and, ultimately, end customers. All of these parties have their own unique ways of doing business, reporting sale and purchase data, and collecting and paying against agreed upon contracts.

Unfortunately, many of the steps and processes carried out to facilitate fuel sales and distribution are archaic, inefficient, and, often, paper-based. Such inefficient ways of doing business cause many parties to engage large teams of personnel to manage the intricate details often involved in fuel sale and distribution. Fuel deal pricing provides a good example of the inefficiencies involved in moving large volumes of natural gas and other fuels.

Typically, pricing fuel deals in the natural gas arena involves manual processes related to gathering fuel index rates, manually computing sales prices across a multitude of fuel sales deals, laboriously factoring in transportation and other tangential costs, and managing for fuel overages and short falls often associated with transportation anomalies, etc. These processes typically involve the efforts of large teams of personnel within organizations who are required to constantly monitor sales deals, set pricing limits for sales people, and track and record fuel deal progress.

While many systems have been developed to facilitate sale and distribution of fuel and other products, commodities, and services in general, no systems developed to date can effectively management the volume of transactions among a wide array of parties to efficiently and effectively get fuel from one place to another. Moreover, existing systems have heretofore not been able to facilitate pricing practices that factor in past fuel deal data across a multitude of prior fuel deals to better drive profit margins in the commodities and brokerage fields.

Accordingly, there exists a serious need to provide systems and methods that enable centralized location and management of fuel deal data, provide for application of pre-determined pricing techniques based on such fuel deal data, facilitate broad-based reporting based on such centrally stored fuel deal data to drive better business practices for parties to fuel deals, and increase productivity and make more efficient fuel sale and distribution practices. The present invention squarely addresses such a need and provides a new and improved systems and methods for facilitating fuel sale and distribution.

SUMMARY OF THE INVENTION

The present invention solves the problems mentioned above with regard to prior systems and methods used to facilitate sale and distribution of fuel to a customer. By squarely addressing the limitations of prior systems and methods, the present invention provides new and improved systems and methods that permit a wide array of users to broadly access a central data store to create and manage fuel deal data. Such new and improved systems and methods further permit the inclusion of pricing processes into existing business processes that are based on prior fuel deal data and which take into account prior prices charged across collections of prior fuel deal contracts.

Accordingly, the present invention provides new and improved systems and methods for facilitating sale and distribution of fuel to a fuel customer. Such systems and methods include and involve a server facility configured to store fuel deal data and to process such fuel deal data to automatically generate pricing data based on the fuel deal data and in accordance with a pre-determined pricing technique. The system and method also include and involve a client facility that is coupled to the server facility via an electronic data network and which is configured to permit a user to enter such fuel deal data and to cause the server facility to store and process the fuel deal data to generate the pricing data. As such, fuel may be sold and distributed to a fuel customer via a fuel distribution system based on the fuel deal data and the automatically generated pricing data.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is described in detail below with reference to the following drawing figures, of which:

FIG. 1 is a timing diagram that depicts process flows within a business process that facilitates sale and distribution of fuel to customers in accordance with a preferred embodiment of the present invention;

FIG. 2 is a system diagram in which client systems can access server system(s) to facilitate sale and distribution of fuel to customers in accordance with the business process illustrated in FIG. 1;

FIG. 3A is an entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3B is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3C is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3D is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3E is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3F is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3G is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3H is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3I is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3J is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3K is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3L is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3M is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 3N is another entity relationship diagram that depicts data relationships among tables and corresponding table entries used to implement the systems that carry out the business process illustrated in FIG. 1;

FIG. 4A is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4B is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4C is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4D is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4E is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4F is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4G is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4H is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4I is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4J is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4K is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4L is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4M is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4N is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4O is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4P is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 4Q is another screen shot of a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1;

FIG. 5A is a flow chart that illustrates the operations carried out to effect a pricing technique and, in particular, one that effectuates a weighted average sales price for fuel deals in accordance with a preferred embodiment of present invention; and

FIG. 5B is the conclusion of the flowchart started in FIG. 5A.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is now described in detail with regard to the drawing figures that were briefly described above.
The systems and methods described herein are illustrative of the exemplary system implemented by way of computer software within a networked data processing environment and which is contained within multiple files housed on the compact disc that is appended to this patent document. Accordingly, the discussion that follows refers to such an exemplary system and those skilled in the art are encouraged to review such appended software in the context of fuel deal management for a complete understanding of the present invention. As noted at the beginning of this patent document, the material contained on the attached compact disc is protected by the Copyright Laws of the United States (17 U.S.C. § 101, *et seq.*) and may not be copied without the express, written authorization of the copyright holder (Highland Energy Corporation). Copyright © 2001, Highland Energy Corporation. All Rights Reserved.

Referring now to FIG. 1, depicted therein is a timing diagram corresponding to the business process carried out within an organization to facilitate sale and distribution of fuel to a customer and which may be set up to utilize the systems and methods provided by the present invention. In particular, FIG. 1, illustrates a monthly or periodic business process involving several phases of operation that are carried out by the systems and methods provided by the present invention including, but not limited to: an availability phase, a bidding phase, a nominating (e.g., gas pipeline nominations, etc.) and routing phase, a third party and sanctioned sales period, a pricing period, an invoicing period, and an accounting period. Together, these periods make up what is referred to herein as a MONTH OF FLOW PROCESS (MFP). The MFP is described next to further illustrate the business operations that are handled by the systems and methods provided by the present invention.

THE MONTH OF FLOW PROCESS (MFP)

Availability Period

During the availability period of the month of flow process, equity contracts for sale and distribution of fuel (those that need to roll from month to month) are established for the next month. These purchase deals define the anticipated volumes by well/meter for each producer. The status for the production month needs are set to 'Availability' at this point. Then, correspondence is transferred (via fax, email and phone conversations) to the various operators/producers in order to confirm the anticipated volumes to be produced.

The anticipated production volume for an entire well/meter is then entered into the system. An entitlement and makeup percentage is used to indicate how much of this volume is actually available to be marketed (represents the owner interest in the production of the well/meter). New deals are setup on the system to represent the new month's purchases. The package description is utilized to assist with easy recognition of volumes, price, etc. (used for identification purposes only). There is a process built within the system to automate the propagation of new deals to the next month (first time into a new month will automatically generate entries for the new month with zero volume amounts). The actual volume stored on the system (at this point) is zero. Only the nominated volumes contain the expected volumes for the production month. These 'nominated" volumes are equal to the estimates provided by the producers and entered into the system during this part of the month of flow. The primary area of the system utilized is the 'Availability' functions (off the system's main menu.)

Bid Week Period

During the bid week of the month of flow process, buyers are found for the volumes that were made available through the availability step described above. The status of the production month of the system needs is set to 'Sales" at this point. By setting the status to 'Sales' all of the price indices will be initially populated and 'seeded' with zero values. Each of the sales is confirmed by a dealmaker and is written up on a deal log sheet. These deal log sheets reflect the pipe/field, meter/well, company, contract, volume, and pricing instructions to support the sale. Prior to completing a deal, the dealmaker will work closely with the volume control group to ensure that appropriate volumes will be available at the well/meter of sale. The dealmakers then complete the deal log sheet entries for the sale and they are transferred to the volume control group for deal creation and entry into the system. Most of the volumes sold during this particular phase are for the equity purchase deals created during the availability period.

Nominating and Routing

During the nominating and routing period of the month of flow process, the volumes to support the sales are routed from the producer's well/meters to the sales wells/meters (primarily to pooling points or field tanks). This process occurs throughout the entire month. When the volumes are routed to specific pool wells/meters, allowances are automatically made by the system for fuel, gathering and transport costs. These costs will net down the actual available volumes that can be applied to the sales deals. When volumes are routed to a pool/tank then these volumes reflect as 'Transport Out volumes'. The volumes then show up as "Transported In' (net fuel) on the receiving meter/well within the system. The primary area with the System utilized during this process is the "Route Volume" menu option within the Routing module (main menu selection of 'Routing' on the System.

Third Party Deals and Sanctioned Sales

During the Third Party Deals and Sanctioned Sales of the month of flow process, the dealmakers complete the third party deals. These deals are typically setup where a specific purchase deal (non equity type) is made to support a specific sales deal. These types of deals will usually have a specific price agreement and volume associated with them. Sanctioned sales represent sales from equity volumes with specific terms (prices, volumes, etc.) to specific sales meters. A sales price for a specific volume is set in advance of the production month with these types of deals. All third party deals are excluded from Weighted Average Sales Price (WASP) calculations as discussed below with regard to FIGS. 5A and 5B (each third party purchase volume exists within its own WASP pool ('None')). All sanction sales deals are included within the WASP calculation but EACH combination sanction sales (purchase-to-sale) will utilize a 'Dedicated' WASP pool during the calculation. In this way, sanction sale costs etc. PLUS netback percentages can be applied. All equity deals combined with the 'Common' WASP pool where costs and prices are aggregated by meter/well based on volume weightings. All deals actually go through the calculation in order determine margins. However, the calculation has been setup to ensure that third party, sanctioned sale and equity pools are calculated without interfering with each other.

Pricing

During the pricing period of the month of flow process, all monthly index based prices are entered immediately when published. These are usually entered just before the beginning of the production month. Daily prices are keyed or otherwise entered throughout the month as they are received. When deals are setup the 'Pricing' function within the System is used to actually calculate a price for the deal ('Price' tab on deal detail screen). Each evening, for example, the 'Price All Deals' function of the System is started. This particular function will re-price all deals for the entire month (Price + WASP calculations). For months in the 'Sales' phase, the nominations are re-priced and recalculated. For months in the 'invoiced' phase, the pipe/field actuals are re-priced and recalculated. In addition, to this periodic process, an option exists within the System to price production months throughout a day, for example. Below, with reference to

FIGS. 5A and 5B, the details related to fuel deal pricing are described. The ability of the present invention to incorporate a pricing technique such as one that is predetermined and implemented as a modular component of a larger software system, represents a significant point of novelty to which the present invention is directed.

Invoicing

During the invoicing period of the month of flow process, invoices for all of the sales for the previous month are produced. This represents the final step of the month within the system. All marketing individuals directly involved with the system for the month (controllers, dealmakers, etc.) are informed that the month is closing out and that invoices are now being produced. The status for the production month is changed to 'Invoiced'. A final nomination calculation is automatically done with the status updated. Accounting is then notified that the month has been completed. Invoicing reports are then run for the month and sent to the buyers by an accounting group, for example. Additional reports may be run (Sales By Pipe/Field, Purchase By Producer, Balancing Reports, Pipeline Statement Comparison Reports, etc.) by the accounting group for historical reference and reconciliation.

Accounting

During the accounting period of the month of flow process, an accounting group creates a revenue and journal entry feed to track receivables within an automated accounting system. This feed is created directly out of the system. Pipe/Field statements begin appearing beginning as early as the 15th of the month. These statements represent volumes (by well/meter) for the previous month. Each accounting analyst is responsible for a specific set of pipe/fields. The volumes from these statements are entered as actuals into the system. A copy of the Pipe/Field statements are sent to the controllers for sign off. Accounting analysts then balance all of the purchase meter routing information for their respective pipe/fields. Accounting analysts then balance all of the sales meters for their respective pipe/fields. Accounting analysts then adjust any route volumes that cross pipe/fields to ensure interconnect balances are synchronized with pipe/field statements. Reconciliation and voucher reports can be run immediately after the production month is promoted to 'Accounting' phase (meaning accounting is finished with the month). These reports can then be sent to producers and/or entered into to accounting system.

AN EXEMPLARY SYSTEM

Referring now to FIG. 2, depicted therein is a diagram of an exemplary system in which client systems can access server system(s) to facilitate sale and distribution of fuel to customers in accordance with the business process (MFP) illustrated in FIG. 1. In particular, system 100 includes both server(s) 102 and client systems 104. Additionally, a database management system and corresponding data store 106 (hereinafter data store 106) is used to store fuel deal data and programs. Servers 102 are configured to be accessed via wide area network connections such as those facilitated via the Internet using open standards based protocols. Client systems 106 are configured with software contained on the appended compact disc to access servers 102 to engage in fuel deal operations such as those described with reference to the month of flow process (MFP) discussed above with regard to FIG. 1.

In FIG. 2, client systems 104 may be configured as desktop computing systems, wireless computing clients, etc. to access servers 102. Such access may be made possible via applications and technology such as dbOvernet TCP/IP Socket Connection Middleware. Furthermore, servers 102 execute common SEServer applications and routines utilizing dbOvernet middleware technology.

Within the processing space of servers 102, a database server system such as Microsoft's SQLServer V.7.03 (a DBMS engine) may be instantiated. Such a database management system may control data store 106 and may be configured in accordance with the present invention to maintain all fuel deal data in accordance with the present invention.

The following discussion further defines an exemplary arrangement for a client-server system implemented in accordance with the present invention:

SERVERS

MS Windows NT 4.0 (SP6) may be chosen as a Network Operating System.

The DBMS may be Microsoft's SQL-Server (V7.0x) – Service Pack 3. All data generated and processed within the context of the present invention is stored in MS SQL-Server database tables. Such data is accessed via direct SQL statements (embedded in Windows applications, stored procedures, forms, and reports). There are several database views that have been setup to access aggregated information (for performance and consistency). In addition, all of the critical calculations and time consuming procedures such as pricing calculations, routing and rollover processes, etc. are written as Transact-SQL stored procedures and are contained on the attached compact disc and are discussed in further detail below in the embedded description-tables found herein.

The SEServer may be a Middleware Server Application. The system database is accessed via middleware software that uses TCP/IP (SEServer/dbOvernet). All databases queried through the system come through this middleware component.

SECrystal (Crystal Reporting Engine Server Application) may be used for server side reporting functions, etc. All reports for the system utilizes a remote Crystal Reporting engine (SECrystal) server. These reports are run and saved on the server for electronic distribution. Crystal Report (V8.0) from Seagate Software is used for this function.

The SEFax (Fax Server Application) may be used for Fax distribution. This server application is responsible for sending out reports via a fax device. This software monitors a specific directory and when a fax file 'shows up' in the directory it will be faxed.

The MAPI Mail Client Software provides Email (like Microsoft Outlook or Outlook Express). The MAPI compliant email service needs to be running on the same machine as the report engine server (SECrystal). This provides the ability to email reports (Correspondence) automatically. Options should be set on this client to automatically check (send/receive) at periodic intervals.

The Adobe Acrobat Reader (Free PDF Viewer) is used to view reports, etc. The server machine that runs the SECrystal reporting server application needs to also have the PDF viewer installed. This is used in order to 'spool' to paper the print jobs.

WEB ACCESS – NETWORK CONNECTIVITY

All functions within the System are available over the Internet (with appropriate security). An individual wishing to log in to the system over the Internet will need to have appropriate application security to log in, the current application executable program (as contained on the attached compact disc) and an ISP account. System administrators will need to furnish access site addresses (e.g., IP addresses, domain names, etc.) to users to address the systems provided by the present invention.

CLIENT SYSTEMS

Client systems may utilize a Client Operating System such as MS-Windows 95/98/Me; MS-Windows NT 4.0/2000. TCP/IP network protocol is required. Access to the server TCP/IP address (either LOCAL address or REMOTE address is required.)

The system typically includes a single .EXE file(s) (plus approximately 8 disk compression and graphics DLL's). The system application require only a single executable with a few DLL's to reside on the client machine. No other client configuration software is required. Upgrades to the client software are automatically done when a user first connects (logs in) through the Internet (on application startup). A version number check will be made if necessary and a new installation program and script are automatically downloaded.

The Adobe Acrobat Reader (FREE PDF view) is used as a reporting system for files saved in the PDF 1.2 format. The default output for all reports on the system is the standard PDF format. This provides for email/electronic storage. In order to view reports this software (or other third party viewer with a file association to .pdf files) needs to exist on the client machine.

The MAPI Mail Client Software is used for electronic mail communications. A MAPI compliant email service needs to be running on the client machine to be able to highlight a report and email it using the client email address list. This software is not required to run the but is required to take advantage of the system's ability to attach reports automatically within an email client.

All client applications are written using DELPHI (V5+) including Delphi 3rd party tools and procedures. Such applications and stored procedures and identified 3rd party tools are further described in the description-tables found below.

DATABASES, AND CORRESPONDING ENTITY RELATIONSHIPS

The various database tables that make up the system have been divided into nine (9) database subject areas. A subject area within this context is simply a logical aggregation of tables that support a particular business or system function. All of the database tables physically reside in the same database, but are not required to so reside. Only the documentation (as described below) has been constructed to illustrate these subject areas. It is also important to note that there are linkages (not documented here) between the various subject areas.

These database subject areas and a description include:

Companies: All company related tables (including company name, contact name, addresses functions, etc.).

Contracts: All contract related tables (including contract provisions, notes, default standard reporting, etc.).

Deals: All deal related tables (includes other costs, deal classes, correspondence, etc.).

Volume Inventory: All volume inventory tables (includes production interests, daily monthly, calculated values, etc.).

Operational: All tables that were created to support the system (software application). These tables include fax queue tables, printer definition tables, system logs, system messages, reporting tables, etc.

Pipes/Fields: All pipe/field and meter/well related tables.

Pricing: All tables within the system that are related to pricing (indices, price descriptions, baskets, etc.).

Routing: All tables within the system that define routes (leg definitions, daily leg rates, monthly leg rates, nom and actual volume routing instructions, etc.).

Security: All security related tables within the system (includes user, logins, passwords, business functions, etc.).

The above-described nine (9) logical database subject areas are next broken down into the actual tables that reside on the attached compact disc. For purposes of brevity, such database subject areas are broken down in the following tables:

<THIS SPACE LEFT BLANK INTENTIONALLY>

Below is an inventory of the various database tables that are utilized by the Energy Management System. This particular inventory indicates the current number of rows (through January 2001), the database (MS SQL Server) and the database subject areas (logical grouping of tables).

Ref #	Table Name	Rows	Database	Subject Area	Description/Comments
	Companies Subject Area				
1.0	Address	1,384	SQL Server	Companies	Contains record entries for each address for all companies and contacts within companies (multiple address types per company and/or contact).
2.0	Company	1,242	SQL Server	Companies	Contains a record entry for each company in the database. Information on this table includes company name, fax, phone and primary address reference identifier.
3.0	Contact_Group	908	SQL Server	Companies	Contains a record entry for each contact group relationship. This is the mechanism for grouping company contact individuals..
4.0	Contact_GroupNames	8	SQL Server	Companies	Contains a record entry for each contact group name.
5.0	ContactFunction	997	SQL Server	Companies	Contains a record entry showing the contact to function relationships for a given company.
6.0	Contacts	3,347	SQL Server	Companies	Contains a record entry for each individual contact in the database. Includes full name, phone, fax, email, title, etc.
	Contracts Subject Area				
10.0	K	1,414	SQL Server	Contracts	This table contains a record entry for each contract within the system. Bank information (ABA), Evergreen indicators, termination date, fixed pricing, etc. type data attributes are stored on these records. Each contract on the system has an associated parent 'company' (on the Company table).
11.0	KNetBack	334	SQL Server	Contracts	This table contains the netback pricing tiers associated with a given contract. The parent table for this entity is the contract table (K). The netback pricing tiers are volume and date influenced.
12.0	Knotes	589	SQL Server	Contracts	This table contains an optional record entry for each contract on the system. If there are no notes associated with a contract then the records are not inserted on the database. This provides the users with a free form area for keeping notes about a contract.
13.0	Kproducts	1,049	SQL Server		This table contains a reference to the products that are available (oil, gas, liquids, etc.) for a given contract. A product has to be associated to a contract before a deal can be setup using that contract for that product.
14.0	KreportDefaults	48	SQL Server		This table contains the entire standard reporting defaults for a particular entity. These reports include invoices, remittance, vouchers, deal confirmations, etc.
15.0	KreportOverrides	0	SQL Server		If a particular contract has its own unique standard reports then a reference to these unique reports is stored in this table for the contract in question.

Ref #	Table Name	Rows	Database	Subject Area	Description/Comments
16.0	Kservices	1,068	SQL Server		This table contains a reference to the services that are available (marketing, end user, pass thru, etc.) for a given contract. A service has to be associated to a contract before a deal can be setup using that contract for that service.
	Deals Subject Area				
20.0	RdealClass	6	SQL Server	Deals	This table is a reference table that indicates the types of deal class options that are available. The context of each class is 0=Purchases. 1=Sales and 2=Both. The description field indicates the possible answers (but the rDealClassA table contains the actual answers that can be applied).
21.0	RdealClassA	23	SQL Server	Deals	This table is a reference table that indicates the possible deal classification options for each of the classifications defined in the rDealClass table.
22.0	RdealClassRules	448	SQL Server	Deals	This table contains record entries for every combination of deal classification answers (rDealClassA table). Each of these classification options can have its own set of calculation rules/etc associated with it.
23.0	Engine_Master	39,149	SQL Server	Deals	This table contains a record entry for each price entry effective date (header record).
24.0	Engine_MasterPrice	79,244	SQL Server	Deals	This particular table contains the individual pricing components associated to a given deal on a given effective date (parent record is on the Engine_Master table). When the user of the Energy Management System enters a price, this is the table that gets updated.
25.0	Package	65,351	SQL Server	Deals	This table contains a record for each deal that has been setup on the system. Start Date, End Date, Deal Name, Contract, Company, etc. are specified on this table.
26.0	PackageCosts	381	SQL Server	Deals	This table contains entries for all 'other costs' associated with a given deal. Each of these 'other costs' will have unique STID's (deal or meter level) and have calculated 'Engine' records automatically generated (when a calculation runs).
27.0	PackageCorrespondence	3,447	SQL Server	Deals	This table contains entries for all of the electronic correspondence between the parties to the deal (deal confirmations, availability reports, remittance detail, vouchers, etc.).
28.0	PriceComponents	19	SQL Server	Deals	This table contain record entries for each component that can be set aside for pricing purposes (on a deal). Examples include 'DAILY INDEX', 'MONTHLY INDEX', 'GATHERING', etc. These tags will be associated to each component of the price to allow for future queries and reporting. In addition, these tags will provide an audit trail of all pricing related information.
29.0	PriceDesc	33,877	SQL Server	Deals	This table contains a record for each deal description (or comments) within the system. These price description records (only 1 per deal) provide the users with a place to put free form text to help describe the price of the deal.
	Volume Inventory Subject Area				

Ref #	Table Name	Rows	Database	Subject Area	Description/Comments
30.0	Engine	280,970	SQL Server	Volume Inventory	This table contains record entries for eacn calculated transaction that the system attaches to volume inventory items. Each transaction has a unique STID (transaction id) that are defined in the Engine_TransactionList table. Indicators on this table determine the disposition of the transaction.
31.0	Engine_TransactionList	36	SQL Server	Volume Inventory	This table contains record entries that define all of the transactions that can be calculated and stored in the Engine table. The STID field is the unique transaction identifier.
32.0	GasInv	159,501	SQL Server	Volume Inventory	This is the primary table were all volumes (nominated and actual) are maintained This table contains the header record entries that shows by month, company, transaction, pipe/field & meter/well the nominated volume and the estimated actual volumes. References to price types, contracts, etc. are stored on each record.
33.0	GasInvD	4,145,617	SQL Server	Volume Inventory	This table contains the detail (DAILY) nominated and estimated actual volumes for the GasInv table.
34.0	ProdInterest	7,999	SQL Server	Volume Inventory	This table contains a record that lists the production interests that are held for a given meter/well and contract (with date effectiveness).
35.0	ProdPkg	4,080	SQL Server	Volume Inventory	This table contains a record that indicates (by month) the contract and the deal ID of a deal that was generated automatically within the 'Availability' (equity purchase deal creation) area of the system.
36.0	ProdSum	39,296	SQL Server	Volume Inventory	This table contains records that indicate (by month and meter/well) the gross mmbtu's and the Btu factors.
37.0	ProdVol	44,187	SQL Server	Volume Inventory	This table contains record entries (by month and meter/well) which show the receipt and delivery mmbtu's per day.
	Operational Subject Area				
40	ApplicationMessages	55,882	SQL Server	Operational	This table contains a 'rolling' 7 day listing of all application messages (such as those that are displayed to the console during a calculation).
41.0	ExceptionCategories	8	SQL Server	Operational	This table contains record entries to hold all of the exception 'reasons' that will be used whenever an exception even occurs. There can be multiple types of exception categories.
42.0	ExceptionList	2,171	SQL Server	Operational	This table contains entries for the actual exception events that get logged by the system. These represent an audit trail of non-normal or error type information. This table is linked to the ExceptionCategories table because each exception event (in this table) requires a reason category.
43.0	LogTable	4	SQL Server	Operational	This table is used for debugging purposes only and is not used in any screens or reports.
44.0	PrinterDef	6	SQL Server	Operational	This table contains a record for each available printer (including driver and port).

Ref #	Table Name	Rows	Database	Subject Area	Description/Comments
45.0	RgasMonth	1,440	SQL Server	Operational	This is a reference table that contains a record for each month from 1/1980 thru 12/2099. In addition, this table also contains the status and status update sequence number for the particular month. This status is used in order to enable/disable certain functions within the Energy Management System throughout the month.
46.0	RGasMonthStatus	1,873	SQL Server	Operational	This represents a historical audit table that will be updated every time the monthly status for a given production month is modified (via triggers on the RgasMonth table). This provides a mechanism of identifying who & when the changes were for the status, over time.
47.0	SEMessages	1,251	SQL Server	Operational	All system messages are stored in this table.
48.0	SEAudit		SQL Server	Operational	This table contains record entries for those events that are deemed 'auditable'. Some examples include 'Login' events, Actualization balancing events, standard report submission events, etc.
49.0	SEImages	2	SQL Server	Operational	This table contains record entries that contain graphic images for the screen and reports used throughout the system.
50.0	SELocations	3	SQL Server	Operational	This table contains record entries that define the server paths (network folder locations) where certain key correspondence items are found. For example (report location, deal correspondence, etc).
51.0	SEProcessingCodeTypes	15	SQL Server	Operational	This table contains the 'Type' codes to the reference table 'SEProcessingCodes'. An example is the type code of 'CONTRCTPRD' which describes a reference code for contract products.
52.0	SEProcessingCodes	143	SQL Server	Operational	This table contains reference codes for various fields used throughout the Energy Management System.
53.0	SERptsExecutedStats	19,117	SQL Server	Operational	This table contains record entries that lists the start and end date and times for all reports that were submitted. This provides statistics on how long to execute/etc.
54.0	SERptsGroupItems	218	SQL Server	Operational	This table contains entries of each specific report that exists within a reporting tab (group) within a specific reporting folder (category).
55.0	SERptsGroups	36	SQL Server	Operational	This table contains a list of all available reporting tabs (groups) within each reporting folder (category).
56.0	SERptsItemDetail	123	SQL Server	Operational	This table contains the list of all available reports within the system.
57.0	SERptsItemParms	657	SQL Server	Operational	This table contains record entries for each report parameter for each report defined to the system. Options exist for substituting a different label name than actual parameter field name.
58.0	SERptsQueue	5,667	SQL Server	Operational	This table contains record entries for all 'submitted' reports (report queue). When reports are automatically removed from the system the record is removed from this queue.
59.0	SERptsQueueDistribute	7,855	SQL Server	Operational	This table contains entries that dictate how to distribute the output of reports from the queue (fax, email, printer, etc.).
60.0	SERptsQueueNotify	276	SQL Server	Operational	This table contains entries that indicate who (and if) individuals or groups have been notified that the report has finished.
61.0	SERptsSchedule	0	SQL Server	Operational	This table contains records that define specific schedules for the running of scheduled reports.

Ref #	Table Name	Rows	Database	Subject Area	Description/Comments
62.0	SERptsScheduledReports	0	SQL Server	Operational	This table contains record entries that define which reports to run as part of specific schedules.
63.0	SERptsScheduledGroups	0	SQL Server	Operational	This table contains 'groups' for scheduling. This provides the ability to assign multiple individuals to a specific group and have the group belong to the schedule.
64.0	SERptsScheduledUserGroups	0	SQL Server	Operational	This is the actual table that contains the members within a schedule group. Each entry in this table defines the group.
65.0	SERptsTablesUsed	896	SQL Server	Operational	This table contains documentation on what tables, views or stored procedures are used within each report.
	Pipes & Fields Subject Area				
80.0	Meter	4,335	SQL Server	Pipes and Fields	This table contains a record entry for each well/meter that has been setup on the system. The pipe/field, name, county and state are stored here.
81.0	MeterNotes	935	SQL Server	Pipes and Fields	This table contains a record for notes pertaining to meters/wells.
82.0	PipeField	372	SQL Server	Pipes and Fields	This table contains a record entry for each pip/field defined on the system. The company and the pipe/field description are stored here.
83.0	MeterRates	3,980	SQL Server	Pipes and Fields	This table contains the entire pressure base, Btu factors by effective date for specific meters/wells.
84.0	MeterAllocations	551	SQL Server	Pipes and Fields	This table contains entries for the allocation information on the meter/well. This includes accounting cross-reference codes (id and deck).
	Pricing Subject Area				
90.0	GCIndex	142,268	SQL Server	Pricing	This table contains record entries by Day for daily index prices AND/OR a single entry for monthly index prices (1st day of month for monthly indices).
91.0	IndexRef	228	SQL Server	Pricing	This represents the master table of all defined price indices within the Energy Management System. One record entry per index exists within this table.
92.0	IndexBaskets	14	SQL Server	Pricing	This table contains a record entry for each index basket established on the system. These index baskets can be associated to sales or purchase deals just as normal indexes are associated to them. Simple averages are calculated with all index items within an index basket.
93.0	IndexBasketLink	36	SQL Server	Pricing	This table contains the actual indices that are currently associated with an index basket. An unlimited number of indices can exist in a basket. A simple average of all the prices within the basket is used.
	Routing Subject Area				
101.0	LegRef	4,226	SQL Server	Routing	This table contains record for each unique transportation leg (meter-to-meter) on the Energy Management System.

Ref #	Table Name	Rows	Database	Subject Area	Description/Comments
102.0	Leg	57,830	SQL Server	Routing	This table contains a record for each active leg within a given month. Nomination and actual rates that the leg utilizes during the month are posted on each record. These rates are used with the actual routing instructions (LegDetail table).
103.0	LegD	0	SQL Server	Routing	This table contains OPTIONAL entries for any daily leg rates that need to be utilized within a given month. Daily rates are checked PRIOR to the monthly rates (on the Leg table) when setting up the actual routing instructions (LegDetail table).
104.0	LegDetail	1,716,695	SQL Server	Routing	This table contains the detail routing instructions for all volumes purchased all the way through the sales points for that particular volume. Nomination AND actual routing instructions are stored for each meter/well that had volume activity during the month. All volumes sold can be tracked back to originating purchase points.
105.0	WASPResovedRouting	34,304	SQL Server	Routing	This table contains record entries that show the pool level calculated totals for all receipt and delivery points within the system. 'Common', 'Dedicated' and 'None' pools are aggregated and the total numbers stored here. Only 'Common' pool volumes and dollars represent the totals from more than one purchase point (shows weighted average pricing based on volumes purchased and/or transported).
	Security Subject Area				
110.0	GCUser	27	SQL Server	Security	This table contains a single record entry per unique user (employee) on the system. The character based (up to 12 character) login ID AND an internal user id (integer) are unique keys to this table.
111.0	GCButton	58	SQL Server	Security	This table contains records that represent the system functions that have specific security rules associated with them on the system. For example a system function of 'DEALS' has been setup in order to define security relationships between users (GCUser table) and this function.
112.0	GCSecurity	1,548	SQL Server	Security	This table stores the relationships between users on the system (GCUser table) and the system function that they have access too (GCButton table). A specific access privilege is stored for each of these relationships (like READ ONLY, READ/UPDATE, READ/UPDATE/DELETE or SUPER).

Referring now to FIGS. 3A-3N, depicted therein are entity relationship diagrams that illustrate data relationships among tables and corresponding table entries used to implement the systems and methods that carry out the business process illustrated in FIG. 1. The database tables used logically categorized above into the above-identified nine (9) subject areas are maintained within data store 106 (FIG. 1), and are included among the files present on the attached compact disc, and are further defined in detail in FIGS. 3A-3N. Those skilled in the art will readily understand the data relationships among relational database tables as shown in FIGS. 3A-3N. Accordingly, for purposes of brevity, further comments about FIGS. 3A-3N have been omitted.

In addition to the tables described and specified in the tables listed above, the following table illustrates an inventory of the various database views that utilized by the systems and methods provided by the present invention.

<THIS SPACE LEFT BLANK INTENTIONALLY>

VIEW DESCRIPTIONS

Below is an inventory of the various database views that are utilized by the Energy Management System:

Ref #	View Name	Description/Comments
1.0	V_SearchDB	Provides a view to search the database stored procedures and triggers for specific text items. Used for assessing the impact of system changes.
2.0	VAccountingRevenueFeed	Database view (3 select UNION) used for creating OGSYS journal and revenue receivable data.
3.0	VCompany	Display of company information (name, address, etc.)
4.0	Vcontact_Accounting	Display the accounting contact for a given company.
5.0	Vcontact_Admin	Display the administrative contact for a given company.
6.0	Vcontact_Control	Display the control contact for a given company.

Ref #	View Name	Description/Comments
7.0	Vcontact_Production	Display the production contact for a given company. This is the contact used for Availability estimates/etc.
8.0	Vcontact_volconfirm	Display the contact responsible for confirming volumes within a given company. This is the contact used for volume confirmations in the 'Availability' phase.
9.0	VcontactFunction	Display a list of all contacts for a given company along with their respective functions (accounting, volume confirmations, etc.)
10.0	VContacts	Display name and addresses for contacts.
11.0	VETID_Dates	Display the engine start, effective and end dates for a given engine transaction id (based on package). This view is used VERY LITTLE because of performance issues.
16.0	VgasInvD_NomChg	Display list of daily volumes where the nomination volumes are different between two successive days.
17.0	VKTermination	Displays specific contract termination information.
18.0	VlegDetail_MeterTotals	Display routing information summarized by meter.
19.0	VlegDetail_PipelineComparison	Display routing information in a format that is used for the pipe/field comparison report. Used for reconciling fuel, gathering, transport, pvr, etc to pipe/field statements.
20.0	VlegDetail_PurchasePointTotals	Display routing information that shows total routing costs/etc for given purchase points (hop 0's).
21.0	VlegDetail_Summary	Displays routing information (summarized) for reporting purposes (purchase meters/wells only).
22.0	VlegDetail_SummarySales	Displays routing information (summarized) for reporting purposes (sales meters/wells only).
23.0	VMeterAllocations	This view is used to list the current meter/well allocations (based on effective date) for each given meter/well. These allocations are the accounting deck and purchaser id information, which can be different from month to month.
24.0	VMeterRates	This view is used to list the current meter/well rates (standard pressure base, pipe/field pressure base, Btu factor, etc.) for each given meter/well. These rates can be different from month to month.
25.0	VOurContact_Accting	Display the current HEC contact for accounting information.
26.0	VOurContact_Prod	Display the current HEC contact for production information.
27.0	VPackage_Info	Display detail list of information concerning a package (includes contacts, names, phones, etc.).
28.0	VPrevGasMonthStuff	Displays current month volume totals versus previous month volume totals.
29.0	VprodConfirmLetters	Display contact information for use with correspondence on production volumes. Specifically used in the confirmation process in the 'Availability' production month phase.
30.0	VprodInterest	Display a list of contracts and meters to confirm the production interests. This is used primarily in the 'Availability' production month phase.
31.0	VRequestProduction	Display list of production interest volume and meter information. This is used primarily in the 'Availability' production month phase and is used when sending out estimate reports to producers.

Once all software and data as described above has been properly installed on one or more server systems 102 and within one or more coupled (networked) client systems 104 as illustrated in FIG. 1, use and operation of the systems and methods provided by the present invention may be commenced. Such operations may be in relation to the general use application (Energy Management System - EMS) or the limited use/user/function application (Producer Control Console – PCC) provided on the attached compact disc. In either case, the present invention facilitates a client-server application environment that includes, among other things, a user interface that is pleasing to users and which permits easy and ready access to system functions and operations. Such a user interface may be a graphical user interface or GUI that is configured to permit a user to engage in window-operations to bring about database operations that affect fuel deal data and the like in accordance with the present invention. Such a GUI is illustrated by way of screen shots (images of computer monitor screens) that are used to permit generation of, manipulation of, reporting of , and all other system operations relating to fuel deals and corresponding fuel deal data.

For example, reference is now made to FIGS. 4A-4Q which illustrate a data processing application running within a client system to facilitate at least some of the operations carried out to effect the business process illustrated in FIG. 1. FIG. 1, for example, represents an opening main menu screen through which a user may select "PERSONAL" operations related to setting up a personal profile to affect user-preferred presentation of data (e.g., name, screen colors, etc.). Additionally, a user may select "PRICE-INDEX" to affect fuel pricing and index related data. A user may select "COMPANY" to control lists of producers, and other related company entities. A user may select other options corresponding to the steps involved and described with regard to the MONTH OF FLOW PROCESS illustrated and described with reference to FIG. 1.

The other screen shots shown in FIGS. 4B-4Q further illustrate specific features of the GUI that has been designed to facilitate the implementation of the systems and methods provided by the present invention. For the purpose brevity, further detailed comments related to such screen shots has been omitted.

SYSTEM IMPLEMENTATION AND FUNCTIONALITY

As noted above, the present invention utilizes stored procedures in the form of database management system procedures and functions which are executed server-side and client-side to facilitate the present invention's systems and methods. Listed in the following tables, is a detailed break-down of all the stored procedures, tools, and modules used to facilitate such systems and methods. The actual source code and instructions contained with in such procedures, functions, and modules is contained on the attached compact disc.

STORED PROCEDURES

Below is an inventory of the various database-stored procedures (procedures and functions) that are utilized by the systems and methods provided by the present invention. Each of the stored procedures and functions are written in the Transact-SQL dialect. All of the stored procedures are prefixed with "usp_" which stands for "User Stored Procedure." This provides an ability to differentiate those procedures bundled with the DBMS versus those created for the systems and methods provided by the present invention:

Stored Procedures

Below is an inventory of the various database-stored procedures (procedures and functions) that are utilized by the Energy Management System. Each of these stored procedures and functions are written in the Transact-SQL dialect. All of the stored procedures are prefixed with "usp_" which stands for "User Stored Procedure". This provides an ability to differentiate those bundled with the DBMS versus those specifically created for the EMS application.

Ref #	Stored Procedure Name	Description/Comments
1.0	Usp_DailyCleanup	This procedure is run everyday and is responsible for any cleanup activities (like rolling aged messages off the ApplicationMessages table).
2.0	Usp_fGetCalcIndex	Retrieves the weighted average price for a given volume item. This routine is invoked during the WASP calculation in order to obtain the price for the meter/well and post it to the Engine database table.
3.0	Usp_fGetIndex	Retrieves the daily or monthly price index for a given day. Used during the pricing calculation routine.
4.0	Usp_fGetIndexBasket	Retrieves and calculates the index amounts for the price lines whenever an index basket price variable has been entered. This particular function will return the average price (simple average) of all indices within the basket for a given month/day.
5.0	Usp_fGetNetbackPercentage	This function will return the actual netback percentage to be used for a given production month and contract. When it calculates the netback it looks at volumes and tier instructions that have been setup on the contract. The number it returns is the netback percentage to utilize. In addition, this routine brings back the specific percentage to use for the product being calculated (gas, liquids, oil, etc.).

09835014.052801

Ref #	Stored Procedure Name	Description/Comments
6.0	Usp_fGetProdInterestID	This routine brings back the production interest information for a particular ownership interest.
7.0	Usp_fGetProdPkg	This procedure brings back the 'deal id' (if one already exists) when posting volumes through the 'Availability' screens. If a deal does not already exist (in the current production month) then a new deal is created and that ID is sent back.
8.0	Usp_fGetWASPIndicator	This function accepts a deal id (package ID) as it's input. It then reads the DealClass table and the rDealClass table(s) to determine if this particular deal should be considered WASPable based on its classification scheme. The return values are either 'None', 'Common' or 'Dedicated'.
9.0	Usp_fGetWaspType	This procedure will send back the WASP type field (GAS, OIL or LIQUIDS) when passed a specific product ID. This procedure is used during the calculation in order to determine which set of netback rules off a contract to use.
10.0	Usp_fIsLastDay	This procedure accepts a date and sends back the last date in a production month.
11.0	Usp_fLastDay	This procedure accepts a date and sends back the last date in a production month.
12.0	Usp_fPipeContactInfo	This procedure, when passed a pipe/field id, will send back the specific contact information requested (like accounting contact, volume contact, etc.).
13.0	Usp_GasDayToGasMonth	This function will return the production month to use for a given production day.
14.0	Usp_GetProductVolumeRound	This routine will return the rounding precision necessary when calculating volume information for specify products (Oil calculates to 2 decimal places, Gas to zero, etc.).
15.0	Usp_LinePrice	This is the actual procedure that will calculate the Engine records for a given deal (volume related STID 8 or 9 type records).
16.0	Usp_message	This routine handles all of the 'progress' messages that are issued during the calculation, rollover, actualization, and etc. type events on the system. This routine will optionally post this information to the ApplicationMessages table for historical reference (audit).
17.0	Usp_pActualize_BalPurchases	This is the main driver routine for Step 2 of 4 of the actualization process.
18.0	Usp_pActualize_BalPurchasesCheck	This routine will check to see if all of the meters/wells on a given pipe/field have been actualized. If not, then it sends back a bad return code. All meters/wells are required to be 'checked' (actualized) prior to balancing of purchase routing points
19.0	Usp_pActualize_BalPurchasesClear	This routine is the actual routine that will adjust all purchase meter imbalances. These imbalances are adjusted forward THROUGH the sales point based on nomination routing instructions (used as a map).
20.0	Usp_pActualize_BalSales	This is the main driver routine for Step 3 of 4 of the actualization process.
21.0	Usp_pActualize_BalSalesCheck	This routine will check to see if all of the purchase meters/wells routing balances (from step 2 of 4) are balanced. If any meter/well on the pipe/field is out of balance then this routine sends back a bad return code. All meters/wells on the pipe/field are required to be 'balanced' prior to balancing of the sales points.
22.0	Usp_pActualize_BalSalesClear	This procedure is the final procedure invoked by the usp_pActualize_BalSales main driver procedure. It is responsible for posting imbalance amounts to the internal clearing purchase or sales deals.
23.0	Usp_pActualize_BalSalesOver	This procedure attempts to reconcile any outstanding balances that result in OVER supplying of volume to a particular sale. Nomination information is used by this routine as a 'road map' on how to allocate this volume.
24.0	Usp_pActualize_BalSalesUnder	This procedure attempts to reconcile any outstanding balances that result in UNDER supplying of volume to a particular sale. Nomination information is used by this routine as a 'road map' on how to allocate this volume.
25.0	Usp_pFillIndex	This procedure will initialize the records within the 'GCIndex' table with daily entries (for daily indices) and monthly entries (for monthly indices). The monthly record entries are only on the first day of the month.
26.0	Usp_pFillIndexSingle	This procedure will populate the 'GCIndex' table with a price index entry for a SINGLE index.
27.0	Usp_pGasInvD_Fill	This routine initially populates the daily volumes on the GasInvD table. These are initially populated with zeros (anytime a meter/well is added to a deal).
28.0	Usp_pGasInvD_NomEOM	This routine is used in the 'Availability' area of the EMS system and is used to take a given volume amount and propagate that volume amount to all days in the month.
29.0	Usp_LogAuditInfo	This routine is used to post record to the audit table within the system.
30.0	Usp_pPackageRevision	This routine is used to increment the revision number field on the deal. Certain types of changes to a deal will automatically increment the revision number on a deal and this update is done through this routine.

Ref #	Stored Procedure Name	Description/Comments
31.0	Usp_pPostClassificationRules	This procedure is executed (usually by triggers on the rDealClass and rDealClassA tables). It can be executed stand-alone. This procedure will ensure that a record is created in the rDealClassRules table for every combination of deal classification codes (dcA values on the rDealClassA table).
32.0	Usp_ProdPush	This routine is used in the 'Availability' phase of EMS and is used to initially populate a particular month with ownership interest information, by meter/well.
33.0	Usp_pPushMeter	This routine is used in the 'Availability' phase of EMS and is used to populate a single meter/well ownership interest to its respective deal (package) and volume inventory item (GasInv/GasInvD).
34.0	Usp_pRouteBuildLegHistory	This routine creates the 'Leg' records for a given meter/well. When a new 'route' (LegRef) is defined on the system then this routine will get invoked to initially seed the 'Leg' table with entries in order to allow routing.
35.0	Usp_pRouteBuildLegHistoryAll	This routine gets invoked when a production month is initially opened to the 'Sales' phase. All ACTIVE meters and legs will have their respective 'Leg' table records populated for that production month by this routine.
36.0	Usp_pRouteCopyLegHistoryActuals	This routine gets invoked when the status of a production month changes from 'Sales' to 'Invoiced'. All nomination routine instructions (in the 'LegDetail' table) are then copied by this routine. This provides the mechanism to have actuals different than noms while preserving the nom instructions.
37.0	Usp_pRoutePostChange	This procedure gets invoked whenever a change to a specific route is requested (i.e. modifications of volumes between hops).
38.0	Usp_pRoutePostDealInfo	This procedure gets invoked to 'seed' the 'LegDetail' table with routing information. This is invoked when new meters/wells are added to deals.
39.0	Usp_pRoutePostDealInfoVols	This procedure gets invoked to populate the specific volumes on each of the 'LegDetail' entries (daily) for deal inventory items.
40.0	Usp_pRoutePostDelete	This procedure gets invoked whenever a deletion is requested on the routing (LegDetail) information.
41.0	Usp_pRoutePostLegRates	This procedure gets invoked in order to post the rates (fuel, pvr, transport, gathering, etc) to each of the 'LegDetail' records in the database. Daily rates (LegD table) overrides monthly rates (Leg table) and this procedure ensures that priority. If a rate gets changed for a leg this routine gets invoked to update all existing routes (LegDetail) records.
42.0	Usp_pRoutePostSale	This procedure gets invoked in order to post volume (route it) to a sales item (in the LegDetail table).
43.0	Usp_pRoutePostTransport	This procedure gets invoked in order to post volume (route it) to a transportation point (in the LegDetail table).
44.0	Usp_pRouteRemoveLegDetails	This routine will remove any/all 'LegDetail' (routing instructions) when a meter/well for a specific deal is removed.
45.0	Usp_pSERPT_GetAdditionalReportInfo	This routine is used by all of the 'standard' reporting procedures to obtain specific report fields needed when running a standard report.
46.0	Usp_pSERPT_PostReportToCorrespondence	This routine will post a 'PackageCorrespondence' table record to a particular deal that is affected by the 'standard' report being run. This routine is called by all standard report routines.
47.0	Usp_pSERPT_PostReportToDistribution	This routine will post a report distribution request to the SERptsQueueDistribute table. This is either a request to 'PRINTER', 'EMAIL' or 'FAX'.
48.0	Usp_pSERPT_PostReportToQueue	This routine is used by all of the standard report routines and will post an actual report request (queue item) to the SERptsQueue table.
49.0	Usp_pSERPT_RunReportAvailConfirms	This routine is responsible for running the 'Availability' confirm reports.
50.0	Usp_pSERPT_RunReportAvailEstimates	This routine is responsible for running the 'Availability' estimate reports.
51.0	Usp_pSERPT_RunReportDealConfirm	This routine is responsible for running the deal confirmation reports (from the deal detail screen on EMS).
52.0	Usp_pSERPT_RunReportInvoice	This routine is responsible for running all standard invoice reports.
53.0	Usp_pSERPT_RunReportRemittance	This routine is responsible for running all standard remittance reports.
54.0	Usp_pSERPT_RunReportVoucher	This routine is responsible for running all standard voucher reports.
55.0	Usp_pSERPT_SetAParameterBoolean	This routine is used by the standard reporting routines and converts Boolean parameters for posting on the report queue (SERptsQueue) table.
56.0	Usp_pSERPT_SetAParameterDate	This routine is used by the standard reporting routines and converts date and date/time parameters for posting on the report queue (SERptsQueue) table.
57.0	Usp_pSERPT_SetAParameterDecimal	This routine is used by the standard reporting routines and converts decimal (number) parameters for posting on the report queue (SERptsQueue) table.
58.0	Usp_pSERPT_SetAParameterInteger	This routine is used by the standard reporting routines and converts integer number parameters for posting on the report queue (SERptsQueue) table.
59.0	Usp_pSERPT_SetAParameterString	This routine is used by the standard reporting routines and converts string parameters for posting on the report queue (SERptsQueue) table.

Ref #	Stored Procedure Name	Description/Comments
60.0	Usp_pSERPT_WhichReport	This routine is used by the standard reporting routines and is responsible for determining WHICH report to use. The default reports are in KreportDefaults table. However, any given contract can override the default (KreportOverrides table).
61.0	Usp_PSPrice	This is the main pricing routine for the volume inventory items (regular purchases and sales).
62.0	Usp_PSPriceAll	This is the main procedure for calculating the prices for a given month on a set of deals (volume inventory pricing, STID 8 & 9). Parameters to this stored procedure dictate the type of price to calculation (Nom or Pipe/Field Actual and Sales versus Purchase, etc.).
63.0	Usp_PSPriceAnyNewInvoicesNeeded	This routine is responsible for assigning new invoice and remittance numbers to the volume inventory table (GasInv). If new meters/wells (volume entries) get entered during the actualization process then this routine will ensure they are assigned unique numbers.
64.0	Usp_PSPriceAssignInvoiceNo	This routine assigns invoice numbers to all sales deals when the production month is promoted to the 'Invoiced' phase.
65.0	Usp_PSPriceAuto	This procedure run everyday and checks for any production month either in the 'Sales' or the 'Invoiced' phase. If any production months are within these phases then this procedure will invoke the calculation routine (usp_psPriceAutoMonth) for them.
66.0	Usp_PSPriceAutoMonth	This is the main driver routine for the calculation of an entire month.
67.0	Usp_PSPriceComponentsCheck	This procedure will automatically insert system generated price components (like WASP or Netback Percentage) to the Engine_Master table. It is invoked by the usp_PSPriceI procedure when calculating prices on a deal for a given month.
68.0	Usp_PSPriceCost	This is the routine that calculates the 'Other Cost' entries and posts calculated results in the Engine table.
69.0	Usp_PSPriceCostAll	This is the main driver routine for looping through all of the 'Other Costs' in a given month and invoking the usp_PSPriceCost routine for each one.
70.0	Usp_PSPriceCreateActualEntries	This procedure copies the pricing entries setup on each deal (Engine_MasterPrice) from nom to actuals.
71.0	Usp_PSPriceMarkActualAdjustments	This procedure gets invoked by the calculation routine to mark any volume inventory item (GasInv) whenever a difference is detected between nominations and actuals.
72.0	Usp_PSPricePopulateEngine	This procedure will populate the Engine table FROM the Engine_Master table. For daily index price entries this procedure will automatically propagate the daily index price to all days of the month where there is a volume (at least until a new pricing entry is found). Only volume entries are populated here (STID 8 & 9).
73.0	Usp_PSPriceTransportAll	This routine calculates all of the transport costs for a given production month. These transport costs (and volumes) are posted in the GasInv (pricetype=3) table and deals are posted (if needed). These deals are tagged with the specific transport contract.
74.0	Usp_PSPriceWASPCalc	Determines and resolves all wasp 'Common' and 'Dedicated' pools. Dedicated pools are sanctioned sales. This is the main driver procedure for the wasp portion of the calculation. Third party (pool = 'None') are also processed within this procedure but not for the intent of obtaining a price for them, totals used primarily for profit margin reporting.
75.0	Usp_PSPriceWASPCalcResolveDriver	This is the main driver component for driving the WASP calculation.
76.0	Usp_PSPriceWASPCalcResolveN	Traces back sales totals from all sales meters back to their originating purchase points. The table updated here is the WASPResolvedRouting table. The 'LegDetail' table is used extensively in this calculation. This is a highly ITERATIVE process.
77.0	Usp_PSPriceWASPCalcResolveSalesN	This procedure creates the entries in the WASPResolvedRouting table and posts original sales volumes and amounts. This is done just prior to the routine that resolves these sales totals back to the purchase points.
78.0	Usp_PSPriceWASPCalcSalesN	Sums all WASPable sales by sales meter into the WASPSalesMeterTotals table.
79.0	Usp_PSPriceWASPClearMonth	This routine runs when a production month is promoted to 'Completed' phase. Any volume inventory items (GasInv and/or GasInvD) or routing items (LegDetail) that contain zeros are removed so that only relevant information is stored in the database for historical purposes.
80.0	Usp_PSPriceWaspDivieOutProceedsN	This procedure is the main procedure that will distribute the proceeds from those deals that have been designated to have their respective proceeds distributed via the 'Financial Overrides' setup on the deal.
81.0	Usp_ProdVolSet	This routine is used in the 'Availability' phase to setup the ownership interest on a particular pipe/field and meter. ProdSum and ProdVol tables for the current production month are populated with this procedure.

Ref #	Stored Procedure Name	Description/Comments
82.0	Usp_ProdVolSetAll	This routine is used in the 'Availability' phase to setup the ownership interest on all pipe/fields and meters. This routine invokes the usp_ProdVolSet routine for each meter/well in the loop.
83.0	Usp_PSRollover	This routine gets invoked when a production month goes from 'Availability' to 'Sales' and is responsible for copying deal information month-to-month.
84.0	Usp_PSRolloverPopActuals3	This routine gets invoked by the usp_PSRollover routine and is responsible for populating noms with previous 3 months actuals numbers (primarily used for Oil).
85.0	Usp_PSRolloverPopNoms	This routine gets invoked by the usp_PSRollover routine and is responsible for populating noms with previous months nom numbers.
86.0	Usp_pStatusChanged	This routine gets invoked anytime the production month status is changed (Initial,Availability,Sales,Invoiced,Accounting,Completed). Other routines are invoked depending on the from and to status for the production month.
87.0	**Usp_w.***	Any stored procedure that begins with Usp_w_ has been setup as a one time only procedure that is used to correct any database items/etc. These procedures can be permanently deleted and have no impact on existing functions within EMS.

Application Software

TECHNICAL SKILL SET REQUIRED

Support and maintenance of the Energy Management System requires the following technical skill set.

Ref #	Skill Set	Used For...
1.0	SQL-Server (Transact SQL)	All data is stored in MS SQL-Server database tables. This data is accessed via direct SQL statements (embedded in windows applications, stored procedures and reports). There are several database views that have been setup to access aggregated information (for performance and consistency). In addition all of the critical calculations and time consuming procedures (like the main EMS calculation, routing and rollover process) are written as Transact-SQL stored procedures (and documented in this manual).
2.0	Delphi (V5 +) (includes Delphi 3rd party tools)	All client applications are written using this particular RAD tool. In addition to knowing the standard components that come with this tool, any of the 3rd party tools (documented in this manual) are used extensively. See the 3rd party tools listed in the 'Tools Utilized' section for more details.
3.0	Crystal Reports (V8.0)	All reporting within EMS is done utilizing this tool from Seagate software.

CLIENT SERVER APPLICATIONS W/TOOLS UTILIZED

This particular section contains the high level documentation relative to the Energy Management System software application. Each item documented is uniquely numbered to aid in reviews and/or future modifications.

Ref #	Item	Response	Comments
1.0	Client Application	Energy Management System	The Energy Management System is written in Delphi 5 (service pack 3 applied). Third party controls and components were used in the development. See other areas of this matrix for 3rd party tools utilized.
2.0	Client Application	Producer Control Center	The Producer Control Center is written in Delphi 5 (service pack 3 applied). Third party controls and components were used in the development. See other areas of this matrix for 3rd party tools utilized. This application provides a restricted view of information specific to the company/contact that is running the application. The data viewed is the same data that is maintained in the EMS system.
3.0	Server Application	Software Experts, Inc. SECrystal (V8.00)	All reporting done within EMS utilizes Crystal reports. This server application runs and stores all output reports for the system. Besides storing an electronic copy of the report, this server can distribute to a printer, fax folder OR an email address if instructed by the EMS application.
4.0	Server Application	Software Experts, Inc. SEFax (V2.00) (outbound faxing)	Some output reports (from SECrystal) are designated to be faxed. This software is responsible for faxing all of the reports that were designated by EMS to be faxed.

Ref #	Item	Response	Comments:
5.0	Server Application	Software Experts, Inc. SEServer (V2.00g) (database request server)	All database requests for the Energy Management System AND the Producer Control Center go through this database server component. This server application typically runs on the same machine as the actual database.
6.0	3rd Party Tool/Library	Adobe Acrobat Reader (V4.0 +)	This free tool is used to view reports from EMS. The default for all reports is to print them to a PDF format. This output format is 'overrideable' by the user when the report is submitted. Other formats like Excel, Word, Text, etc. are also supported.
7.0	3rd Party Tool/Library	Seagate Software Crystal Reports (V8.00)	All reports are written using the Crystal reporting tool from Seagate Software). In addition, the report server (SECrystal) utilizes the main Crystal reporting FREE runtime libraries to run these reports for all EMS client requests.
8 0	3rd Party Tool/Library	Dalco Technologies DbOvernet (V2..00)	Delphi VCL components that provide internet (TCP/IP) access. The SEServer application utilizes this middleware.
9.0	3rd Party Tool/Library	TurboPower Software Asynch Pro (V3.04)	The SEFax fax server application utilizes this 3rd party Delphi VCL component list for sending and/receiving faxes. The SECrystal reporting server application uses this library to write out 'fax ready' files.
10.0	3rd Party Tool/Library	TurboPower Software Orpheus (V3.08)	Many of the online screens for all client and server applications utilize the Orpheus controls for screen grid lists, combo boxes, etc. The server applications were written with this tool set also.
11.0	3rd Party Tool/Library	TurboPower Software SysTools (V3.02)	Many of the online screens for all client and server applications utilize the SysTools components for string manipulations, spawning tasks, etc.
12.0	3rd Party Tool/Library	Woll2Woll Software InfoPower 2000.17	Many of the online screens for all client and server applications utilize these controls for screen grid lists, combo boxes, etc. The server applications were written with this tool set also.
13.0	3rd Party Tool/Library	Inner Media.Software Dynazip (V4.00)	These are Delphi software components that are for compression/decompression of files to and from the server. This is used by both the client and server applications.
14.0	3rd Party Tool/Library	Public Domain TEmail (V2.10)	This is a Delphi software component and is used by the client and server applications. It is responsible for the email interface.
15.0	3rd Party Tool/Library	TMS Software TwebUpdate (V1.00)	This is a Delphi software component that provides for 'over the internet' automatic software upgrades. The client applications each utilize this component.
16.0	3rd Party Tool/Library	Skyline Software, Inc. ImageLib Suite (V5.00)	These are Delphi software components that provide for graphic images displayed within the application. In addition, this software provides scanner input capabilities.

CLIENT APPLICATIONS, MODULE LIST/DESCRIPTIONS

This particular section contains the high level documentation relative to each software application module within the Energy Management System. Each item documented is uniquely numbered to aid in reviews and/or future modifications. The application reference listed below will either indicate EMS (Energy Management System) and/or PCC (Producer Control Center). This shows the level of interoperability between these two client applications. All of these modules are written in Delphi (Object Pascal, (Visual)).

Ref #	Module Name	Module Type	Application	Description/Comments
1.0	DBAddress	Data Module	EMS PCC	This module contains all of the database communication components for the Address ('Company and Contact Addresses') table.
2.0	DBCommonDatabase	Data Module	EMS PCC	This module is responsible for setting all of the common properties for all other data modules within the system. Prior to invoking a query, all other database modules will invoke methods within this module to set communication ports, maximum number of records, etc. This module also stores the actual user id and contains methods for accessing this field.

Ref #	Module Name	Module Type	Application	Description/Comments
3.0	DBCommonFileOperations	Data Module	EMS PCC	This module handles all of the 'flat file' operations (compressing/decompressing/etc.) that is involved with the applications. Any temporary files that need to be created are also controlled by this data module.
4.0	DBCompany	Data Module	EMS PCC	This module contains all of the database communication components for the Company ('Company Information') table.
5.0	DBContactFunction	Data Module	EMS PCC	This module contains all of the database communication components for the ContactFunction ('Roles within their respective companies that contacts play") table.
6.0	DBContacts	Data Module	EMS PCC	This module contains all of the database communication components for the Contacts ('Individual contacts within companies') table.
7.0	DBContactGroup	Data Module	EMS PCC	This module contains all of the database communication components for the ContactGroup (Links contacts to groups they may be affiliated with) table.
8.0	DBContact_GroupNames	Data Module	EMS	This module contains all of the database communication components for the Contact_GroupNames (table contains a record for each group within the system) table.
9.0	DBEngine	Data Module	EMS	This module contains all of the database communication components for the Engine (contains transaction records for each volume inventory transaction item associated with the deal) table.
10.0	DBEngine_Master	Data Module	EMS	This module contains all of the database communication components for the Engine_Master (User enterable pricing area 'header' record) table.
11.0	DBEngine_MasterPrice	Data Module	EMS	This module contains all of the database communication components for the Engine_MasterPrice (User enterable pricing area 'detail' records (price tags)) table.
12.0	DBEngine_TransactionList	Data Module	EMS	This module contains all of the database communication components for the Engine_TransactionList (transaction descriptions) table.
13.0	DBExceptionCategories	Data Module	EMS PCC	This module contains all of the database communication components for the ExceptionCategories ('Reasons for Exceptions') table.
14.0	DBExceptionList	Data Module	EMS PCC	This module contains all of the database communication components for the ExceptionList ('Actual Exception Events) table.
15.0	DBGasInv	Data Module	EMS	This module contains all of the database communication components for the GasInv (Volume Inventory 'header') table.
16.0	DBGasInvD	Data Module	EMS	This module contains all of the database communication components for the GasInvD (Volume Inventory Daily 'detail') table.
17.0	DBGCButton	Data Module	EMS PCC	This module contains all of the database communication components for the GCButton ('Business Functions') security table.
18.0	DBGCIndex	Data Module	EMS PCC	This module contains all of the database communication components for the GCIndex (Daily & Monthly Price Indices) table.
19.0	DBGCSecurity	Data Module	EMS PCC	This module contains all of the database communication components for the GCSecurity (Security Authorizations) for the applications.
20.0	DBGCUser	Data Module	EMS PCC	This module contains all of the database communication components for the GCUser (User Profiles) table within the applications.
21.0	DBImages	Data Module	EMS	This module contains all of the database communication components for the SEImages (company logos, etc.) table within the application.
21.0	DBIndexBasketLink	Data Module	EMS PCC	This module contains all of the database communication components for the IndexBasketLink (Links actual indices to a particular basket) table within the application.
22.0	DBIndexBaskets	Data Module	EMS PCC	This module contains all of the database communication components for the IndexBaskets (Grouping of indices to be used in a 'simple' averaging calculation) table within the application.

Ref #	Module Name	Module Type	Application	Description/Comments
23.0	DBIndexRef	Data Module	EMS PCC	This module contains all of the database communication components for the IndexRef (Each price index within the system contains a record entry here) table within the application.
24.0	DBK	Data Module	EMS	This module contains all of the database communication components for the K (Contracts table within the application).
25.0	DBKNetBack	Data Module	EMS	This module contains all of the database communication components for the KNetBack (Contracts Netback Percentage Tiers) table within the application.
26.0	DBKNotes	Data Module	EMS	This module contains all of the database communication components for the KNotes (Contract Notes) table within the application.
27.0	DBKProducts	Data Module	EMS	This module contains all of the database communication components for the KProducts (products that are available within contracts) table within the application.
28.0	DBKReportDefaults	Data Module	EMS	This module contains all of the database communication components for the KReportDefaults (standard report defaults) table within the application.
29.0	DBKReportOverrides	Data Module	EMS	This module contains all of the database communication components for the KReportOverrides (standard report overrides for a contract) table within the application.
30.0	DBKServices	Data Module	EMS	This module contains all of the database communication components for the KServices (services that are available within contracts) table within the application.
31.0	DBLeg	Data Module	EMS	This module contains all of the database communication components for the Leg (available routes and rates for the production month) table within the application.
32.0	DBLegD	Data Module	EMS	This module contains all of the database communication components for the LegD (available DAILY routes and rates for the production) table within the application.
33.0	DBLegDetail	Data Module	EMS	This module contains all of the database communication components for the LegDetail (specific routing instructions for all volumes purchased and sold) table within the application.
34.0	DBLegRef	Data Module	EMS	This module contains all of the database communication components for the LegRef (master list of routes and rates) table within the application.
35.0	DBLocations	Data Module	EMS PCC	This module contains all of the database communication components for SELocations (locations) table within the application.
36.0	DBMessages	Data Module	EMS PCC	This module contains all of the database communication components for the SEMessages (system messages) table within the application.
37.0	DBMeter	Data Module	EMS	This module contains all of the database communication components for the Meter/Well table within the application.
38.0	DBMeterAllocations	Data Module	EMS	This module contains all of the database communication components for the MeterAllocations (ownership interests in volume from a meter/well) table within the application.
39.0	DBMeterNotes	Data Module	EMS	This module contains all of the database communication components for the MeterNotes table within the application.
40.0	DBMeterRates	Data Module	EMS	This module contains all of the database communication components for the MeterRates (pressure base, Btu factor, etc. from a meter/well) table within the application.
41.0	DBMiscQueries	Data Module	EMS PCC	This module contains all of the miscellaneous queries that were created to enable views of various tables within the application.
42.0	DBPackage	Data Module	EMS	This module contains all of the database communication components for the Package (Deals) table within the application.
43.0	DBPackageCorrespondence	Data Module	EMS	This module contains all of the database communication components for the PackageCorrespondence (electronic copies of documents associated with deals) table within the application.

Ref #	Module Name	Module Type	Application	Description/Comments
44.0	DBPackageCosts	Data Module	EMS	This module contains all of the database communication components for the PackageCosts ('Other Costs' associated with deals) table within the application.
45.0	DBPipeField	Data Module	EMS	This module contains all of the database communication components for the PipeField (Pipe/Field information) table within the application.
46.0	DBPriceComponents	Data Module	EMS	This module contains all of the database communication components for the PriceComponents (tags to associate to each portion of a price) table within the application.
47.0	DBPriceDesc	Data Module	EMS	This module contains all of the database communication components for the PriceDesc (Deal free form price description) table within the application.
48.0	DBPrinterDef	Data Module	EMS	This module contains all of the database communication components for the PrinterDef (printer definitions) table within the application.
49.0	DBProcessingCodes	Data Module	EMS PCC	This module contains all of the database communication components for the SEProcessingCodes (reference code description) table within the application.
50.0	DBProcessingCodeTyes	Data Module	EMS	This module contains all of the database communication components for the SEProcessingCodeTypes (type codes that classify sets of reference codes) table within the application.
51.0	DBProducerMessage	Data Module	PCC	This module contains all of the database communication components for the ProducerMessage (dynamic messages posted to producers) table within the application.
52.0	DBProdInterest	Data Module	EMS	This module contains all of the database communication components for the ProdInterest (Availability royalty interests) table within the application.
53.0	DBProdPKG	Data Module	EMS	This module contains all of the database communication components for the ProdPKG (Availability deal ID to ProdVol cross reference) table within the application.
54.0	DBProdSum	Data Module	EMS	This module contains all of the database communication components for the ProdSum (Availability summary totals by meter/well) table within the application.
55.0	DBProdVol	Data Module	EMS	This module contains all of the database communication components for the ProdVol (Availability detail owner interest totals by meter/well) table within the application.
56.0	DBrDealClass	Data Module	EMS	This module contains all of the database communication components for the rDealClass (All of the available deal classifications) table within the application.
57.0	DBrDealClassA	Data Module	EMS	This module contains all of the database communication components for the rDealClassA (all possible answers available to the deal class rules (rDealClass table)) table within the application.
58.0	DBrDealClassRules	Data Module	EMS	This module contains all of the database communication components for the rDealClassRules (all rules associated with every combination of deal classification) table within the application.
59.0	DBrGasMonth	Data Module	EMS PCC	This module contains all of the database communication components for the rGasMonth (an entry exists here for every possible month within the system, with status information) table within the application.
60.0	DBRptsControl	Data Module	EMS PCC	This module represents the main driver module for submitting reports.
61.0	DBRptsExecutedStats	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsExecutedStats (Execution statistics for reports) table within the application.
62.0	DBRptsGroupItems	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsGroupItems (List of reports available within each tab/folder) table within the application.
63.0	DBRptsGroups	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsGroups (List of all tabs within each reporting folder) table within the application.

Ref #	Module Name	Module Type	Application	Description/Comments
64.0	DBRptsItemDetail	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsItemDetail (List of specific reports available throughout all folders and tabs) table within the application.
65.0	DBRptsItemParms	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsItemParms (List of all report parameters available to each specific report) table within the application.
66.0	DBRptsQueue	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsQueue (actual report submission queue) table within the application.
67.0	DBRptsQueueDistribute	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsQueueDistribute (report distribution instructions area) table within the application.
68.0	DBRptsQueueNotify	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsQueueNotify (report notification instructions area) table within the application.
69.0	DBRptsSchedule	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsSchedule (report schedule definition area) table within the application.
70.0	DBRptsScheduledReports	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsScheduledReports (reports belonging to schedule definition area) table within the application.
71.0	DBRptsScheduleGroups	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsScheduleGroups (report schedule groups definition area) table within the application.
72.0	DBRptsScheduleUserGroups	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsScheduleUserGroups (user list belonging to a specific schedule group definition area) table within the application.
73.0	DBRptsTablesUsed	Data Module	EMS PCC	This module contains all of the database communication components for the SERptsTablesUsed (tables, views and stored procedures used by each report area) table within the application.
74.0	DBStoredProcedures	Data Module	EMS PCC	This module contains all of the database communication components for accessing and invoking all stored procedures and functions on the application. Each of these procedures are setup as methods within this class and this particular class acts as a common wrapper for invoking these DB procedures.
75.0	RTCrystalDriverParseMemo	Business Rules	EMS PCC	This module contains all of the string parsing routines used to store reporting parameters, formulas and selection criteria.
76.0	RTDBAddress	Business Rules	EMS PCC	All business rules and edits associated with the application addresses (Address table) are within this particular module.
77.0	RTDBCompany	Business Rules	EMS PCC	All business rules and edits associated with the application companies (Company table) are within this particular module.
78.0	RTDBContactFunction	Business Rules	EMS PCC	All business rules and edits associated with the application contact function (ContactFunction table) are within this particular module.
79.0	RTDBContacts	Business Rules	EMS PCC	All business rules and edits associated with the application contacts (contacts table) are within this particular module.
80.0	RTDBContact_Group	Business Rules	EMS PCC	All business rules and edits associated with the application contact group relationships (ContactGroup table) are within this particular module.
81.0	RTDBContact_GroupNames	Business Rules	EMS	All business rules and edits associated with the application contact group names (Contact_GroupNames table) are within this particular module.
82.0	RTDBEngine	Business Rules	EMS	All business rules and edits associated with the application engine pricing transaction (Engine table) are within this particular module.

Ref #	Module Name	Module Type	Application	Description/Comments
83.0	RTDBEngine_Master	Business Rules	EMS	All business rules and edits associated with the application engine pricing entry (Engine_Master table) are within this particular module.
84.0	RTDBEngine_MasterPrice	Business Rules	EMS	All business rules and edits associated with the application engine pricing components (w/price tags) entry (Engine_MasterPrice table) are within this particular module.
85.0	RTDBEngine_TransactionList	Business Rules	EMS	All business rules and edits associated with the application engine transaction master list (Engine_TransactionList table) are within this particular module.
86.0	RTDBExceptionCategories	Business Rules	EMS PCC	All business rules and edits associated with the application exception categories (ExceptionCategories table) are within this particular module.
87.0	RTDBExceptionList	Business Rules	EMS PCC	All business rules and edits associated with the application exception list (ExceptionList table) are within this particular module.
88.0	RTDBGasInv	Business Rules	EMS	All business rules and edits associated with the application volume inventory transaction header (GasInv table) are within this particular module.
89.0	RTDBGasInvD	Business Rules	EMS	All business rules and edits associated with the application volume inventory transaction detail daily (GasInvD table) are within this particular module.
90.0	RTDBGCButton	Business Rules	EMS PCC	All business rules and edits associated with the application business functions (GCButton table) are within this particular module.
91.0	RTDBGCIndex	Business Rules	EMS PCC	All business rules and edits associated with the application price indices (GCIndex table) are within this particular module.
92.0	RTDBGCSecurity	Business Rules	EMS PCC	All business rules and edits associated with the application security authorizations (GCSecurity table) are within this particular module.
93.0	RTDBGCUser	Business Rules	EMS PCC	All business rules and edits associated with the application users (GCUser table) are within this particular module.
94.0	RTDBImages	Business Rules	EMS	All business rules and edits associated with the application graphic images (SEImages table) are within this particular module.
95.0	RTDBIndexBasketLink	Business Rules	EMS PCC	All business rules and edits associated with the application index price basket link (IndexBasketLink table) are within this particular module.
96.0	RTDBIndexBaskets	Business Rules	EMS PCC	All business rules and edits associated with the application index price baskets (IndexBaskets table) are within this particular module.
97.0	RTDBIndexRef	Business Rules	EMS PCC	All business rules and edits associated with the application price index master list (IndexRef table) are within this particular module.
98.0	RTDBK	Business Rules	EMS	All business rules and edits associated with the application contracts (K table) are within this particular module.
99.0	RTDBKNetBack	Business Rules	EMS	All business rules and edits associated with the application contract netback pricing tiers (KNetBack table) are within this particular module.
100.0	RTDBKNotes	Business Rules	EMS	All business rules and edits associated with the application contract free form note area (KNotes table) are within this particular module.
101.0	RTDBKProducts	Business Rules	EMS	All business rules and edits associated with the application contract products area (KProducts table) are within this particular module.
102.0	RTDBKReportDefaults	Business Rules	EMS	All business rules and edits associated with the application contract standard report defaults area (KReportDefaults table) are within this particular module.
103.0	RTDBKReportOverrides	Business Rules	EMS	All business rules and edits associated with the application contract standard report overrides area (KReportOverrides table) are within this particular module.
104.0	RTDBKServices	Business Rules	EMS	All business rules and edits associated with the application contract services area (KServices table) are within this particular module.

Ref #	Module Name	Module Type	Application	Description/Comments
105.0	RTDBLeg	Business Rules	EMS	All business rules and edits associated with the application leg (monthly) area (Leg table) are within this particular module.
106.0	RTDBLegD	Business Rules	EMS	All business rules and edits associated with the application leg (daily) area (LegD table) are within this particular module.
107.0	RTDBLegDetail	Business Rules	EMS	All business rules and edits associated with the application leg detail (main routing) area (LegDetail table) are within this particular module.
108.0	RTDBLegRef	Business Rules	EMS	All business rules and edits associated with the application leg master list area (LegRef table) are within this particular module.
109.0	RTDBLocations	Business Rules	EMS PCC	All business rules and edits associated with the application locations (SELocations table) are within this particular module.
110.0	RTDBMessages	Business Rules	EMS PCC	All business rules and edits associated with the application messages (SEMessages table) are within this particular module.
111.0	RTDBMeter	Business Rules	EMS	All business rules and edits associated with the application meters (Meter table) are within this particular module.
112.0	RTDBMeterAllocations	Business Rules	EMS	All business rules and edits associated with the application meter ownership allocations (MeterAllocations table) are within this particular module.
113.0	RTDBMeterNotes	Business Rules	EMS	All business rules and edits associated with the application meter comment areas (MeterNotes table) are within this particular module.
114.0	RTDBMeterRates	Business Rules	EMS	All business rules and edits associated with the application meter rate areas (MeterRates table) are within this particular module.
115.0	RTDBPackage	Business Rules	EMS	All business rules and edits associated with the application deals (Package table) are within this particular module.
116.0	RTDBPackageCorrespondence	Business Rules	EMS	All business rules and edits associated with the application deal correspondence (PackageCorrespondence table) are within this particular module.
117.0	RTDBPackageCosts	Business Rules	EMS	All business rules and edits associated with the application deal 'Other Costs' (PackageCosts table) are within this particular module.
118.0	RTDBPipeField	Business Rules	EMS	All business rules and edits associated with the application pipes/fields (PipeField table) are within this particular module.
119.0	RTDBPriceComponents	Business Rules	EMS	All business rules and edits associated with the application price components (PriceComponents table) are within this particular module.
120.0	RTDBPriceDesc	Business Rules	EMS	All business rules and edits associated with the application deal pricing free form text area (PriceDesc table) are within this particular module.
121.0	RTDBPrinterDef	Business Rules	EMS	All business rules and edits associated with the application printer definitions (PrinterDef table) are within this particular module.
122.0	RTDBProcessingCodes	Business Rules	EMS PCC	All business rules and edits associated with the application processing codes (SEProcessingCodes table) are within this particular module.
123.0	RTDBProcessingCodeTypes	Business Rules	EMS	All business rules and edits associated with the application processing code types (SEProcessingCodeTypes table) are within this particular module.
124.0	RTDBProdInterest	Business Rules	EMS	All business rules and edits associated with the application 'Availability' royalty interests (ProdInterest table) are within this particular module.
125.0	RTDBProdPKG	Business Rules	EMS	All business rules and edits associated with the application 'Availability' deal to ProdVol cross-reference (ProdPKG table) are within this particular module.
126.0	RTDBProdSum	Business Rules	EMS	All business rules and edits associated with the application 'Availability' monthly meter summary (ProdSum table) are within this particular module.
127.0	RTDBProdVol	Business Rules	EMS	All business rules and edits associated with the application 'Availability' monthly ownership volume (ProdVol table) are within this particular module.

Ref #	Module Name	Module Type	Application	Description/Comments
128.0	RTDBrDealClass	Business Rules	EMS	All business rules and edits associated with the application deal classification options (rDealClass table) are within this particular module.
129.0	RTDBrDealClassA	Business Rules	EMS	All business rules and edits associated with the application deal classification answers (rDealClassA table) are within this particular module.
130.0	RTDBrDealClassRules	Business Rules	EMS	All business rules and edits associated with the application deal classification wasp rules (rDealClassRules table) are within this particular module.
131.0	RTDBrGasMonth	Business Rules	EMS PCC	All business rules and edits associated with the application production month (rGasMonth table) are within this particular module.
132.0	RTDBRptsExecutedStats	Business Rules	EMS PCC	All business rules and edits associated with the application execution statistics for reporting (SERptsExecutedStats table) are within this particular module.
133.0	RTDBRptsGroupItems	Business Rules	EMS PCC	All business rules and edits associated with the application tab items for reporting (SERptsGroupItems table) are within this particular module.
134.0	RTDBRptsGroups	Business Rules	EMS PCC	All business rules and edits associated with the application tabs for reporting (SERptsGroups table) are within this particular module.
135.0	RTDBRptsItemDetail	Business Rules	EMS PCC	All business rules and edits associated with the application report files used for reporting (SERptsItemDetail table) are within this particular module.
136.0	RTDBRptsItemParms	Business Rules	EMS PCC	All business rules and edits associated with the application report file parameters used for reporting (SERptsItemParms table) are within this particular module.
138.0	RTDBRptsQueue	Business Rules	EMS PCC	All business rules and edits associated with the application report submission queue used for reporting (SERptsQueue table) are within this particular module.
139.0	RTDBRptsQueueDistribute	Business Rules	EMS PCC	All business rules and edits associated with the application report queue distribution options used for reporting (SERptsQueueDistribute table) are within this particular module.
140.0	RTDBRptsQueueNotify	Business Rules	EMS PCC	All business rules and edits associated with the application report queue submission notifications used for reporting (SERptsQueueNotify table) are within this particular module.
141.0	RTDBRptsSchedule	Business Rules	EMS PCC	All business rules and edits associated with the application report schedules used for reporting (SERptsSchedule table) are within this particular module.
142.0	RTDBRptsScheduledReports	Business Rules	EMS PCC	All business rules and edits associated with the application report schedule actual reports used for reporting (SERptsScheduledReports table) are within this particular module.
143.0	RTDBRptsScheduleGroups	Business Rules	EMS PCC	All business rules and edits associated with the application report schedule groups used for reporting (SERptsScheduleGroups table) are within this particular module.
144.0	RTDBRptsScheduleUserGroups	Business Rules	EMS PCC	All business rules and edits associated with the application report schedule users (in groups) used for reporting (SERptsScheduleUserGroups table) are within this particular module.
145.0	RTDBRptsTablesUsed	Business Rules	EMS PCC	All business rules and edits associated with the application report tables used for reporting (SERptsTablesUsed table) are within this particular module.
146.0	RTMessageStackClient	Business Rules	EMS PCC	This particular module is responsible for maintaining the current list of messages that will be displayed to the user. This module will provide for the storing of up to 50 messages (in memory tables) in between enter button or mouse clicks. This provides for any/all error messages concerning a specific event to be displayed at once versus one at a time.
147.0	FmAbout	Form	EMS PCC	This form provides descriptive information about the application (version number, copyright notice, email and website support links, etc).

Ref #	Module Name	Module Type	Application	Description/Comments
148.0	FmActualizePurchases	Form	EMS	This form provides the method for performing (Step 2 of 4) of the actualization process within EMS.
149.0	FmActualizeSales	Form	EMS	This form provides the method for performing (Step 3 of 4) of the actualization process within EMS.
150.0	FmAddressDetail	Form	EMS	This form provides for the updating of addresses for contacts and companies. The table that gets updated behind the scenes is the Address table.
151.0	FmAddressList	Form	EMS	This form provides a list of all available addresses that have already been setup for a company. Options on this form include an ability to change, add or delete address lines from the list.
152.0	FmBusinessFunctionsDetail	Form	EMS	This form provides for the updating of the business functions that are available within the Energy Management System AND the Producer Control Center. The table that gets updated (behind the scenes) is the 'GCButton' table.
153.0	FmBusinessFunctionsList	Form	EMS	This form provides a list of all available business functions that are currently within the Energy Management System AND the Producer Control Center. Options exist here to add, change and delete business functions. Eacn of these business functions represent areas within the application for setting system security.
154.0	FmCommon	Form	EMS PCC	This form provides for all of the common properties used by ali forms. This form can be accessed via the main menus by selecting system properties. All of the color schemes, graphic images, etc. that are used by the system are included on this form. At runtime, all other forms within the system will invoke public methods within this form to set their respective screen fields.
155.0	FmCompanyDetail	Form	EMS	This form provides the mechanism for updating detail information pertaining to a specific company. This includes identification of a primary company address.
156.0	FmCompanyList	Form	EMS	This form provides a grid list of all companies that are currently stored on EMS. Options on this form include extensive lookup and tab options.
157.0	FmContactDetail	Form	EMS	This form provides the form for updating detail information about a contact at a particular company. This includes group memberships, functions, etc.
158.0	FmContactFunctionDetail	Form	EMS	This form provides the mechanism for associating a contact within a company to a specific job function at that company (i.e. Accounting, production, etc.).
159.0	FmContactGroupDetail	Form	EMS	This form provides the mechanism for creating or updating contact groups on the system.
160.0	FmContactGroupList	Form	EMS	This form lists all available contact groups on the system. Options on this form include the ability to add, change or delete a contact group.
161.0	FmContactList	Form	EMS	This for lists all contacts within all companies. Options on this form include an ability to add, change or delete a specific contact (with appropriate security). In addition, there are extensive data lookup capabilities.
162.0	fmContactSecurityAuth	Form	EMS	This form provides for the entry of external company security authorization rules (i.e. Enabling access to Producer Control Center, etc.).
163.0	FmContractDetail	Form	EMS	This form represents the detail form for entering contract specific information (netback pricing information, contract name, terms, provisions, etc.).
164.0	FmContractList	Form	EMS	This form provides a grid list of all existing contracts on the system. Options exist on this form to add, change or delete a contract. This form also includes extensive lookup and company letter tab's for searching all contracts.
165.0	FmDailyPrices	Form	PCC	This form shows the graphs of the revenue detail information on the Producer Control Center.
166.0	FmDealClassificationUpdates	Form	EMS	This form provides the mechanism for changing any calculation rules associated with a given combination of deal classification codes. The WASP inclusion indicator is stored on this table.
167.0	fmDealCorrespondenceDetail	Form	EMS	This form provides an entry form for attaching electronic correspondence to a deal.

Ref #	Module Name	Module Type	Application	Description/Comments
168.0	FmDealCostsEntryDetail	Form	EMS	This form provides for the entry of 'Other Costs' associated with a particular deal.
169.0	FmDealDetail	Form	EMS	This is the main detail form that shows all of the information relative to a deal.
170.0	FmDealEntryNew	Form	EMS	This form represents a popup box that is displayed when a new deal has been requested. This box prompts the user for the type of deal (purchase or sale) and what product and service it is applicable toward.
171.0	FmDealList	Form	EMS	This form provides a listing of all 'Purchase' or 'Sales' deals within a given month on a grid. Options exist on this screen to add, change or delete a deal.
172.0	FmDealPrice	Form	EMS PCC	This is the form that is used whenever a user wants to calculate the prices for a given volume within a given month. The only options on this form are to 'Price All' and only for those production months and volumes that are applicable (based on monthly status).
173.0	FmDealPriceEntryDetail	Form	EMS	This is the main form for entering deal price information within the Energy Management System. The primary underlying tables that get updated include Engine_Master and Engine_MasterPrice.
174.0	FmException	Form	EMS PCC	This form is invoked whenever a system exception occurs within the system. In order to complete the exception a particular user must have a 'Super ID' for the function and he/she must provide an exception reason with a description.
175.0	FmExceptionCategoriesDetail	Form	EMS	This form provides for a detail update screen to update reason code information for a given type of exception.
176.0	fmExceptionCategoriesList	Form	EMS	This form provides a listing grid of all reason code exceptions for a given type of exception.
177.0	FmGraphicViewer	Form	EMS	This form provides an ability to view graphic images and/or scan in graphic images from a scanner. These images can be attached to a deal.
178.0	fmGroupMemberDetail	Form	EMS	This form represents the detail form for associating a contact as a member of a specific group.
179.0	FmImagesDetail	Form	EMS	This form represents the detail form used for posting updates to the application graphic images (logo's, etc.).
180.0	FmImagesList	Form	EMS	This form provides a list of all graphic images (logos) that are currently stored in the system.
181.0	FmIndexBasketDetail	Form	EMS	This form provides a detail update screen to update index price basket information.
182.0	FmIndexBasketLinkDetail	Form	EMS	This form provides a detail update form to allow for the updating of index links to particular baskets.
183.0	FmIndexBasketList	Form	EMS	This form provides a listing grid of all index baskets on the system.
184.0	FmLegDailyDetail	Form	EMS	This form provides the detail rate information associated with a daily leg rate (which overrides the monthly rate when setup on EMS).
185.0	FmLegDailyList	Form	EMS	This form provides a listing of all daily rates that may be setup for a particular leg.
186.0	FmLegDetail	Form	EMS	This form provides the detail rate information associated with the a given leg, on a given production month within the system. Both nomination and actual rate information is available.
187.0	FmLegHistory	Form	EMS	This form provides a historical list of all monthly leg rates that have been established for a given leg.
188.0	FmLegList	Form	EMS	This form provides a list of all legs on the system. Options exist from this screen to select and change (modify) the specific rate information about a leg.
189.0	FmLegMonthlyView	Form	EMS	This form represents a 'view' form that provides a read-only view of all volumes transported in, out, sold and/or on balance for a specific meter.
190.0	FmLegPurchaseLinkMonthlyView	Form	EMS	This form represents a 'view' form that provides a read-only view of all the purchase deals (volumes) that have been attributed to a selected sale.
191.0	FmLegPurchaseLinkView	Form	EMS	This form represents a 'view' form that provides a read-only view of all purchases linked to a specific sale on a given day.

Ref #	Module Name	Module Type	Application	Description/Comments
192.0	FmLegPurchasePointView	Form	EMS	This form represents a 'view' form that provides a read-only view of the originating (hop 0) information for any given volume that is displayed on the routing screen(s).
193.0	FmLegRoute	Form	EMS	This is the main routing screen. Options exist on this screen to select pipe/fields, days, noms or actuals, etc. With appropriate security a person can transport and/or sell volume through this panel.
194.0	FmLegSale	Form	EMS	This form is used as a confirm form for posting volume balances to a sale.
195.0	FmLegSalesView	Form	EMS	This form represents a 'view' form that provides a read-only view of all sales that exist on a given pipe/field for either a single day or an entire month.
196.0	FmLegTransport	Form	EMS	This form is used as a confirm form for transporting volumes to other meters (pools). Options also exist on this form to selectively override transport, gathering, pvr or fuel rates associated with the transport.
197.0	FmLegChange	Form	EMS	This form is used whenever a request is made to change the instructions (either volume or rates) on an existing transport OR sale route item.
198.0	fmLegDelete	Form	EMS	This form is used whenever a routed volume (either transported to a pool or posted to a sale) has been requested to be deleted.
199.0	FmLocationsDetail	Form	EMS	This form provides a detail update form to allow for the updating of location information. These location entries are used throughout the system (versus hardcoding locations within the software).
200.0	fmLocationsList	Form	EMS	This form provides a list form to allow for showing the location information. These location entries are used throughout the system (versus hardcoding locations within the software).
201.0	fmLogin	Form	EMS PCC	This is the common login form used by the application(s). It provides the mechanism for authenticating users or company contacts upon entry into the system.
202.0	fmLoginChange	Form	EMS	This form provides the users of the system with the ability to change their login passwords.
203.0	fmLookup	Form	EMS PCC	This form provides a standard lookup dialog that allows for queries to be run for nearly all other list forms within the system. Most list screens provide a lookup button (binoculars) that will invoke this form.
204.0	fmMessageBox	Form	EMS PCC	This form displays all system messages used within the system. This particular form gets utilized by nearly all other form on the system. The messages displayed by this form include all ERROR, CONFIRMATIONAL, INFORMATIONAL and IN-PROCESS oriented messages.
205.0	fmMeterAllocationsDetail	Form	EMS	This form provides for an entry screen for entering allocation companies and accounting cross reference deck codes for a given meter/well and effective date.
206.0	FmMeterDetail	Form	EMS	This form provides for a detail update form on meter/well information within the system.
207.0	fmMeterList	Form	EMS	This form provides for a list form of all meters/wells within the system.
208.0	fmMeterRatesDetail	Form	EMS	This form provides for an entry screen for entering rates (pressure base, Btu factor, pipe/field pressure base, etc.) for a given meter/well on a specific effective date.
209.0	FmMeterRevenue	Form	PCC	This form provides a meter/well form that shows graphic representation of calculated volumes and prices.
210.0	FmMeterTotalsView	Form	EMS	This form provides a 'view' which is a read-only view of all the meter totals (actualized versus not actualized) for an entire month). A specific deal OR all deals within a month can be viewed through this form.
211.0	FmMonthlyStatusDetail	Form	EMS	This form provides a screen for updating the detail production month status information. This is where users will go to change the status for each production month (depending on security level of the user).

Ref #	Module Name	Module Type	Application	Description/Comments
212.0	FmMonthlyStatusList	Form	EMS	This form provides a grid list of all monthly status information (by status). Options exist here to invoke the detail update screen to update monthly status information (with appropriate security).
213.0	fmNetBackTierDetail	Form	EMS	This form provides the detail form for updating the netback pricing tiers for a given contract. These tiers are referenced (for all WASP classified deals) during the pricing function.
214.0	FmOGISFeeds	Form	EMS	This form provides an entry form for specifying the parameters used to create the OGIS journal entry and revenue receivable accounting feeds. The actual text files are created from this form.
215.0	FmPickACompany	Form	EMS PCC	This form provides a common mechanism for displaying a list of companies to a user and having one of them selected and carried back to the requesting form.
216.0	FmPickAContact	Form	EMS	This form provides a common mechanism for displaying a list of contacts to a user and having one of them selected and carried back to the requesting form.
217.0	FmPickAContract	Form	EMS	This form provides a common mechanism for displaying a list of contracts to a user and having one of them selected and carried back to the requesting form.
218.0	FmPickADeal	Form	EMS	This form provides a common mechanism for displaying a list of deals to a user and having one of them selected and carried back to the requesting form.
219.0	FmPickADealMeter	Form	EMS	This form provides a common mechanism for displaying a list of deal meters to a user and having one of them selected and carried back to the requesting form.
220.0	FmPickALeg	Form	EMS	This form provides a common mechanism for displaying a list of leg (monthly routes) to a user and having one of them selected and carried back to the requesting form.
221.0	FmPickALegRef	Form	EMS	This form provides a common mechanism for displaying a list of LegRef (master routes) to a user and having one of them selected and carried back to the requesting form.
222.0	FmPickALegSale	Form	EMS	This form provides a common mechanism for displaying a list of sales points available for routing to a user and having one of them selected and carried back to the requesting form.
223.0	FmPickAMeter	Form	EMS	This form provides a common mechanism for displaying a list of meters/wells to a user and having one of them selected and carried back to the requesting form.
224.0	FmPickAPipeline	Form	EMS	This form provides a common mechanism for displaying a list of pipe/fields to a user and having one of them selected and carried back to the requesting form.
225.0	fmPickAReport	Form	EMS	This form provides a common mechanism for displaying a list of reports to a user and having one of them selected and carried back to the requesting form.
226.0	FmPipeDetail	Form	EMS	This form provides the detail update form for updating pipe/field information on the system.
227.0	fmPipelineActuals	Form	EMS	This is the main form used for enter actual volumes for meters/wells on the system. The form includes a calculator function for propagating the volumes across all days for the highlighted meter/well.
228.0	fmPipeList	Form	EMS	This form provides the list form to show all pipe/fields currently defined within the system. Options exist on this form to add, update or delete a pipe/field.
229.0	FmPriceComponentsDetail	Form	EMS	This form provides the screen for updating the detail 'price tags' that have been setup on the system. These price tags allow us to identify the various portions of a sale or purchase price.
230.0	FmPriceComponentsList	Form	EMS	This form provides a grid list of all price components (tags) that have been setup on the system.

Ref #	Module Name	Module Type	Application	Description/Comments
231.0	fmPriceIndexUpdates	Form	EMS	This form provides a list of all prices for the daily Index Prices. When entering this form the default date is set to the current date. When prices are being entered on 'Mondays' there is a 'copy to previous weekend' button which will allow for all prices to be propagated back to the previous weekend. Monthly index prices are entered on day 1 only for a given month.
232.0	FmPriceIndicesDetail	Form	EMS	This form provides a screen for updating the price index information on the database (IndexRef table). This includes display order, name, etc.
233.0	fmPriceIndicesList	Form	EMS	This form provides an 'updateable' grid list that shows all price indices on the system. Options exist here to invoke the add/update function (fmPriceIndicesDetail).
234.0	fmPricesByIndexList	Form	EMS PCC	This form provides a graphic and tabular view of index prices for a given month.
235.0	FmPrinterDetail	Form	EMS	This form provides a detail entry form for updating the printer information stored on the system.
236.0	fmPrinterList	Form	EMS	This form provides a list form that shows all printers currently defined on the system.
237.0	FmProcessingCodesDetail	Form	EMS	This form provides the detail form for updating a given set of reference (processing codes).
238.0	FmProcessingCodesList	Form	EMS	This form provides the list form for showing all of the processing codes. Options exist on this form to add, update or delete a given code.
239.0	FmProcessingCodesPick	Form	EMS	This form provides an ability to 'pick' a particular reference code and send it back to the form that invoked the screen.
240.0	FmProcessingCodeTypesDetail	Form	EMS	This form provides the detail form for updating a given set of processing code types (types of reference codes).
241.0	fmProcessingCodeTypesList	Form	EMS	This form provides the list form for showing all of the processing code types. Options exist on this form to add, update or delete a given type.
242.0	FmProdVolCofirms	Form	EMS	This form provides the mechanism for recognizing volumes that were returned by producers. In addition, options exist on this form to send out producer confirmations.
243.0	FmProdVolHist	Form	EMS	This form provides a history list of royalty and makeup percentage interests, by owner, for a given meter/well.
244.0	FmProdVolList	Form	EMS	This form provides the mechanism for entering initial volumes (expected availability) from producers. Option exist on this form to send out producer availability estimate reports.
245.0	FmReportDefaultsDetail	Form	EMS	This form provides a detail screen for setting up the default reports that will be used by entity, product and service on the system. These reports include invoices, vouchers, remittance, etc.
246.0	FmReportDefaultsList	Form	EMS	This form provides a list screen for showing all of the default reports that are setup by entity, product and service on the system. These reports include invoices, vouchers, remittance, etc.
247.0	FmReportOverridesDetail	Form	EMS	This form provides a detail screen for setting up the override reports that will be used by entity, product and service on the system ASSOCIATE TO A SPECIFIC CONTRACT. These reports include invoices, vouchers, remittance, etc.
248.0	FmReportsList	Form	EMS PCC	This is the primary form used for displaying a reporting folder. Within this folder are all of the reporting 'tabs' that are available. Within each tab are all of the specific reports that can be run. A submission, and view button are available here.
249.0	FmReportsParaemeters	Form	EMS PCC	This is the form that is used when entering the various parameters when a report is submitted. Defaults are automatically supplied and the parameters are listed in a grid list format.
250.0	fmReportsView	Form	EMS PCC	This is the main view form for viewing all of the submitted reports. Options exist to view the reports specifically submitted by a user OR to view the reports that were submitted by the scheduler.

Ref #	Module Name	Module Type	Application	Description/Comments
251.0	fmSecurityAuthDetail	Form	EMS	This form represents the form for establishing and updating security authorizations between users and business functions within the Energy Management System. Options exist here to allow for users to have NO ACCESS, READ ONLY, READ/UPDATE, READ/UPDATE/DELETE or SUPER access to a particular area of application.
252.0	fmSecurityAuthList	Form	EMS	This form provides a listing of all security authorizations that are set for each user on the Energy Management System. Options exist on this form to add, update and delete specific security authorizations for any given user of the system.
253.0	FmsRptsInvoice	Form	EMS	This is the primary form used for submitting standard invoice reports.
254.0	FmsRptsRemittance	Form	EMS	This is the primary form used for submitting standard remittance reports.
255.0	fmsRptsVoucher	Form	EMS	This is the primary form used for submitting standard voucher reports.
256.0	FmTransactionDetail	Form	EMS	This form provides for the entry of 'Other Cost' transactions within EMS. Once these transactions are setup in the system, then they can be attached to deals and calculations will be done against them.
257.0	FmTransactionList	Form	EMS	This form provides a list of all the 'Other Cost' transactions that have been setup on the system.
258.0	fmUserProfilesDetail	Form	EMS	This form represents the creation and update form for all users on the Energy Management System. This form provides an administrator with the ability to change name, password, title, default printer, etc. for all users on the system.
259.0	fmUserProfilesList	Form	EMS	This form provides a listing of all users that are capable of using the Energy Management System. Options exist on this form to add, update or delete a specific user.
260.0	fmGasControlMainMenu	Form	EMS	This form represents the main menu for the Energy Management System. All menu options, speed buttons, etc are stored on this form. This particular form is also responsible for invoking the methods to establish a connection and set the form screen attributes (based on user preferences).
261.0	frmProducerControlCenterMain	Form	PCC	This form represents the main menu for the Producer Control Center. All menu options, speed buttons, etc are stored on this form. This particular form is also responsible for invoking the methods to establish a connection and set the form screen attributes (based on user preferences).

APPLICATION (CLIEN-SIDE) SOFTWARE

The table that follows contains the high-level documentation related to the systems and methods provided by the present invention and, in particular, those sub-functions and applications that run client-side in the context of the present invention. In the table that follows, the terms EMS and PCC are used to differentiate (as described above), between a full use application system and a limited use/user/function application system that are provided by the present invention. The actual source code for such application software is contained among the files found on the attached compact disc.

PRICING AND PRICING TECHNIQUES

So far in the aforementioned detailed discussion the present invention, it has been assumed that the particular pricing techniques may be employed to price one or more fuel deals automatically. The present invention certainly permits fuel deals to be priced based on a variety of factors germane to the energy field. Additionally, the systems and methods provided by the present invention permit fuel deals to be priced automatically, in batch or otherwise, based on pricing techniques which are modularized and which are carried out automatically based on prior or other collections of fuel deals and other fuel deal data. Accordingly, teams of sales personnel can have deals priced based on company specifications to meet margin requirements, etc.

One such technique implemented as a modularized process capable of pricing one or more fuel deals in accordance with the present invention is referred to as the WASP technique which stands for the Weighted Average Selling Price technique. WASP permits one or more fuel deals (usually a collection) to be priced to meet organization pricing targets (and margin requirements) based on computed average sales prices across collections of fuel deals. The WASP technique and its supporting computer software are contained herein for purposes of example to illustrate the novelty of having a system that can incorporate a substitutable pricing technique (algorithm) into a business process like or similar to the one depicted in and discussed in regard to FIG. 1.

The WASP Calculation

This particular section contains information on the calculation that occurs to price deals. In the context of the present invention, it is envisioned that there are three situations that can trigger a pricing calculation:

1. The price calculation can be submitted at any time by individuals with appropriate security using the System online pricing screen (see FIGS. 4A-4Q). Only those production months in a 'Sales' (nomination recalculated) or 'Invoiced' (actual recalculated) status can be submitted through this screen;
2. When the status for a production month goes from 'Sales' to 'Invoiced' a final nomination is performed. In addition, when the status of a production month goes from 'Invoiced' to 'Accounting' a final actuals calculation is performed. These production month status 'promotions' occur through the EMS online screens (by individuals with an appropriate level of security); and
3. Each evening, for example, all production months that are in either the 'Sales' or 'Invoice' status will have a calculation cycle run for them. This

calculation begins at approximately 8:00 CST, for example. This ensures that all variables (price index entries, volumes, routing instructions, etc.) that could influence the price of a given set of deals are recalculated and presented as current, the first thing in the morning.

The entire calculation process is comprised entirely of MS SQL-Server Transact-SQL stored procedures. The 'flow' of the calculation can be described with reference to the following six (6) stages:

Stage 1. **Sales Deal Calculations**

Calculate all sales deals first (all pools and deal classifications). This is done because knowing the sales prices (by pool) is required for the following purchase deal calculations.

Stage 2 **WASP Deal Preparation**

This particular stage simply prepares the WASPResolvedRouting table with initial sales pool total dollars and volumes. This is the primary table that is used when repeatedly (such as via iteration) tracing all volumes from the sales point back to originating purchase points.

Stage 3 **Purchase Deal 'None' Pool (3rd Party) Calculations**

All third party purchase deals (belonging to the 'None' (pool) are calculated first. The reason for this is because of the potential that some of these deals having Financial Overrides that are to be distributed to either a 'Common' WASP pool OR to a specific deal. By doing these calculations first, the profit gain or loss (for the financial overrides) can be determined and posted to the appropriate place in the WASPResolvedRouting table.

Stage 4 **<u>Purchase Deal 'Dedicated' Pool (Sanctioned Sales) Calculations</u>**

All sanctioned sales purchase deals are now calculated. The price for these purchases is driven based on a weighted average basis of the sales meters. Sanctioned sale purchase exist in their own pool ('Dedicated') so that no other purchases volumes (and sales of those volumes) will impact the price calculated. Netback percentages are applied.

Stage 5 **<u>Purchase Deal 'Common' Pool (Equity) Calculations</u>**

All equity deals are then calculated. The price for these purchases is driven based on a weighted average basis of the sales meters. All purchases that are classified as 'equity' will share in pricing and costing (weighted). The pricing is based on the 'common' body. Any given purchase deal classified as equity could potentially impact the price that other purchase deals (in the 'common' pool) calculates. Netback percentages are applied.

Stage 6 **<u>Transportation Costs</u>**

This stage of the calculation aggregates all of the transport volumes throughout the month to special transport deals and volume inventory items.

Each of the aforementioned stages of the calculation are invoked from a stored procedure called **usp_PSPriceAutoMonth**. FIGS. 5A and 5B illustrate the process flows corresponding to these 'stages' and the flow of the stored procedures (discussed above) invoked during the calculation. The ordering of these procedures can be tied back to the stages just described above. Actual WASP calculation routines are listed below to aid the reader to completely understand the nature using a predetermined pricing technique in accordance with the present invention.

Weighted Average Sales Price Calculation Routines

The following software routines implement a weighted average sales pricing technique that may be incorporated within a computing environment such as within a server-side processing system to facilitate fuel deal pricing in accordance with a preferred embodiment of the present invention. Accordingly, in the context of the instant invention, the following routines provide a predetermined pricing technique for pricing fuel deals based on past, present, or future deals, or combinations thereof. The following routines are found among the files contained on the attached compact disc, and also have been commented to assist those of ordinary skill in the art understand the details related to actual implementation.

```
/* Microsoft SQL Server - Scripting                  */
/* Server: IS101                                              */
/* Database: EMS                                              */
/* Creation Date 02/13/2001 4:08:41 PM                        */

CREATE PROCEDURE usp_fGetIndex(
                                        @GasMonthX DATETIME,
                                        @GasDayX DATETIME,
                                        @IX VARCHAR(15),
                                        @IndexValuexx DECIMAL(19,6) OUTPUT
                                        )
AS
/*
*****************************************************************************************
Name: usp_fGetIndex

Description: Get the most recent index value for a specified price index.

Inputs:

GasMonthx - Gas month for lookup
```

```
GasDayx - Preferrable gas day used for lookup
Ix - Index id
IndexValuexx - return index value

History:

11/07/2000 JAMIE Modifications to convert from Watcom-SQL to
Transact-SQL.

******************************************************************************************
*/
BEGIN
SELECT @IndexValuexx = 0
/*
*********************************************
* First get the maximum gas day that
* has been entered for this index
* id in this particular month.
*********************************************
*/
SELECT @GasDayX=(SELECT Max(GasDay) FROM GCIndex WHERE GasMonth=@GasMonthX AND
GasDay<=@GasDayX AND IndexID=@IX AND IndexVal<>0)
/*
*********************************************
* Now get the index value for that
* day.
*********************************************
*/
SELECT @IndexValuexx = IndexVal  FROM GCIndex WHERE GasMonth=@GasMonthX AND GasDay=@GasDayX
AND IndexID=@IX
END


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_fGetIndexBasket(
                                        @GasMonthX DATETIME,
                                        @GasDayX DATETIME,
                                        @IndexBasketIDX VARCHAR(15),
                                        @IndexValuexx DECIMAL(19,6) OUTPUT
                                              )
AS
BEGIN
/*
******************************************************************************************
Name:      fGetIndexBasket

Description: This function will get the index basket amount for the specified
month and date.  This function will return a simple average of all the non zero
components within the index for the month and day.

Inputs: GasMonthX (current gas month), GasDayX (day within month) and
IndexBasketIDX (IndexBasket unique identifier).

Outputs: Simple averaged price for the index basket.

History

xx/xx/xx (?) CHIP Original Creation.

04/29/99 JAMIE Modified for WASP 2.10 Release. Structure changes
made to the Engine and Engine_Master tables. In
addition, all documentation added.  This particular
portion of the system required extensive changes
due to the need to store a nom and actual number
```

```
and because all price components are now stored
off the Engine_MasterPrice table (STID's 8 and 9).

11/08/2000 JAMIE Converted to transact-sql.
*********************************************************************************
*/
/*
*******************************************
* Declare all exceptions, cursors
* and local variables that will be
* utilized by this procedure.
*******************************************
*/
DECLARE IndexBasketLink_Cursor CURSOR LOCAL FORWARD_ONLY STATIC FOR
        SELECT indexID FROM IndexBasketLink WHERE IndexBasketID=@IndexBasketIDX

DECLARE @yTotalPrice DECIMAL(19,6)
DECLARE @yTotalIndices INTEGER
DECLARE @yTotalPriceInterim DECIMAL(19,6)
DECLARE @yIndexID VARCHAR(12)
/*
*******************************************
* Initialize all fields here...
*******************************************
*/
SELECT @yTotalPrice=0
SELECT @yTotalIndices=0
SELECT @IndexValuexx=0
/*
*******************************************
* Loop through all of the indices within
* the index basket. Obtain the price
* information.
*******************************************
*/
OPEN IndexBasketLink_Cursor
FETCH NEXT  FROM IndexBasketLink_Cursor INTO @yIndexID
WHILE @@FETCH_STATUS = 0
        BEGIN
                EXECUTE usp_fGetIndex @GasMonthX,@GasDayX,@yIndexID,@yTotalPriceInterim OUTPUT
                IF @yTotalPriceInterim<>0
                        BEGIN
                        SELECT @yTotalPrice=@yTotalPrice+@yTotalPriceInterim
                        SELECT @yTotalIndices=@yTotalIndices+1
                        END
                FETCH NEXT  FROM IndexBasketLink_Cursor INTO @yIndexID
        END
CLOSE IndexBasketLink_Cursor
DEALLOCATE IndexBasketLink_Cursor
/*
*******************************************
* Take the simple average of the totals
* here...
*******************************************
*/
IF (@yTotalPrice<>0) AND (@yTotalIndices<>0)
        BEGIN
                SELECT @IndexValuexx=(@yTotalPrice/@yTotalIndices)
        END
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO
```

```
SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_fGetNetbackPercentage(
                                        @PIDx INTEGER,
                                        @GasMonthx DATETIME,
                                        @TypeNetbackx VARCHAR(12),
                                        @WhichPricex INTEGER,
                                        @yNetbackPercentage DECIMAL(19,8) OUTPUT
                                        )
AS
BEGIN
/*
*****************************************************************************************
Name: usp_fGetNetbackPercentage

Description: This function will return the netback percentage that should be applied
to a particular deal, for a particular month.  This netback percentage is based on the
percentage setup at the contract level for the deal in question.  These percentages
at the contract level (KNetback table) are tiered.  There are two methods of deriving
the percentage.

Method 0  (All or nothing) - With this method the average daily volume for the month
will be used to find the appropriate tier (also based on effective date).  The netback
percentage to use will be the FIRST tier setup on the contract whose average daily volume
does not exceed the total for the gas month on this package.  All gas volume for the
month will use this same percentage (all or nothing).

Method 1  (Accumulating) - With this method the resulting end percentage that will be
used is based on volumes within each tier (they are weighted based on their respective
volumes.  The netback percentage that is calculated is based on the wieghted average of
all percentages, across all tiers using volumes that were applied.

This particular function will work with Nomination (WhichPricex = 0) and Actual
(WhichPricex = 1) volumes.  In addition, this procedure can return both 'GAS' and/or
'OIL' (and or any other) netback (via the TypeNetbackx parameter).


was sent as an input parameter.  The WASP indicator is based on the combination of
deal classifications that have been established for this deal.  The default indicator
is 'N' (ie if classification information can't be found/etc.).  All combinations of
deal classifications should be setup.

Inputs:

PIDx (package ID)
GasMonthx (Gas Month)
TypeNetbackx (type of netback percentage)
WhichPricex (0=Nominations, 1=Actuals)

Outputs:

A single percentage to be applied to the price, representing the netback.

History:

05/13/99 JAMIE Original Creation.

07/22/99 JAMIE  Modified to check for a floor amount
and return that amount if it is greater
than the calculated amount.

09/02/1999  JAMIE Modified to sum volumes either across DEAL, CONTRACT
or COMPANY when determining the correct tier.

08/21/2000 JAMIE Modifications to only sum volumes within the same
product (across entities and services).

11/08/2000 JAMIE Converted to Transact-SQL
```

```
************************************************************************************
*/
/*
*******************************************
* Declare all exceptions, cursors
* and local variables that will be
* utilized by this procedure.
*******************************************
*/
DECLARE @zRound INTEGER
DECLARE @zEntityCID VARCHAR(12)
DECLARE @zKProductID INTEGER
DECLARE @zKServiceID INTEGER

DECLARE @tmpEndDate DATETIME
DECLARE @tmpMaxEffective DATETIME
DECLARE @tmpDaysInPeriod INTEGER
DECLARE @tmpVolumeTotal DECIMAL(19,2)
DECLARE @tmpAccumulatingTotal DECIMAL(19,2)
DECLARE @tmpPrevBand DECIMAL(19,2)
DECLARE @tmpCurrBand DECIMAL(19,2)
DECLARE @tmpBandTotal DECIMAL(19,2)
DECLARE @tmpBandWeightPerc DECIMAL(19,8)
DECLARE @tmpAccumulatingPrice DECIMAL(19,8)

DECLARE @yNetbackMethod INTEGER
DECLARE @yNetbackTierLevel VARCHAR(10)
DECLARE @yAveragePerDay DECIMAL(19,2)
DECLARE @yDailyTotal DECIMAL(19,2)
DECLARE @yeffective DATETIME
DECLARE @ymaxvollevel DECIMAL(19,2)
DECLARE @ynetprice DECIMAL(19,8)
DECLARE @ynetpricefloor DECIMAL(19,8)
DECLARE @yKID INTEGER
DECLARE @yCID VARCHAR(12)
/*
*******************************************
* Get netback method information off the
* contract. The default will be all or
* nothing (most common). However, this
* should always be found on the contract.
*
* 0 = All or Nothing
* 1 = Accumulating
*
* Also, this area of the code sets the
* default for the netback to zero.
*
* In addition, go and get the default
* netback tier level off the contract
* in order to know at what level to
* summarize the volumes when
* performing the calculation. The
* default is 'DEAL' if it can't be found
* or if one is not specified.
*******************************************
*/
SELECT @yNetbackPercentage=0
SELECT @yNetbackMethod=ISNULL((SELECT tier FROM K WHERE KID=(SELECT KID FROM package WHERE
PKG=@PIDx)),0)
SELECT @yNetbackTierLevel=ISNULL((SELECT NetbackTierLevel FROM K WHERE KID=(SELECT KID FROM
package WHERE PKG=@PIDx)),'COMPANY')
SELECT @yKID=ISNULL((SELECT KID FROM package WHERE PKG=@PIDx),0)
SELECT @yCID=ISNULL((SELECT CID FROM package WHERE PKG=@PIDx),'')
/*
*******************************************
* Get the entity, product and service
* information off the deal table. There
* has to be a value on the deal (package)
* table for each of these...
*********************************************
*/
SELECT @zEntityCID=ISNULL((SELECT K.EntityCID FROM Package,K WHERE PKG=@PIDx and
K.KID=Package.KID),'')
SELECT @zKProductID=ISNULL((SELECT KProductID FROM Package WHERE PKG=@PIDx),0)
SELECT @zKServiceID=ISNULL((SELECT KServiceID FROM Package WHERE PKG=@PIDx),0)
/*
*********************************************
* Now calculate the average volume of
* gas per day that this particular
* package has on the system. Remember to
* use the WhichPrice parameter to determine
* which volume to get.
* 0=(Nominated Volume)
* 1=(pipeline actual volume)
*********************************************
*/
EXECUTE usp_fLastDay @GasMonthx,@tmpEndDate OUTPUT
SELECT @tmpDaysInPeriod=(DATEDIFF(day,@GasMonthx,@tmpEndDate) + 1)
IF @WhichPricex=0
          BEGIN
                    IF @yNetbackTierLevel='DEAL'
                              BEGIN
                                        SELECT @tmpVolumeTotal=ISNULL((SELECT SUM(Nom) FROM
GasInv WHERE PKG=@PIDx),0)
                              END
                    IF @yNetbackTierLevel='CONTRACT'
                              BEGIN
                                        SELECT @tmpVolumeTotal=ISNULL((SELECT SUM(GasInv.Nom)
FROM GasInv,Package
                                                  WHERE GasInv.GasMonth=@GasMonthx AND
GasInv.DBCR=0 AND GasInv.PriceType=1 AND GasInv.KID=@yKID
                                                            AND Package.PKG=GasInv.PKG AND
Package.KProductID=@zKProductID),0)
                              END

                    IF @yNetbackTierLevel='COMPANY'
                              BEGIN
                                        SELECT @tmpVolumeTotal=ISNULL((SELECT SUM(GasInv.Nom)
FROM GasInv,Package
                                                  WHERE GasInv.GasMonth=@GasMonthx AND
GasInv.DBCR=0 AND GasInv.PriceType=1 AND GasInv.CID=@yCID
                                                            AND Package.PKG=GasInv.PKG AND
Package.KProductID=@zKProductID),0)
                              END
          END
IF @WhichPricex=1
          BEGIN
                    IF @yNetbackTierLevel='DEAL'
                              BEGIN
                                        SELECT @tmpVolumeTotal=ISNULL((SELECT SUM(PipelineActuals)
FROM GasInv WHERE PKG=@PIDx),0)
                              END
                    IF @yNetbackTierLevel='CONTRACT'
                              BEGIN
                                        SELECT @tmpVolumeTotal=ISNULL((SELECT
SUM(GasInv.PipelineActuals) FROM GasInv,Package
                                                  WHERE GasInv.GasMonth=@GasMonthx AND
GasInv.DBCR=0 AND GasInv.PriceType=1 AND GasInv.KID=@yKID
                                                            AND Package.PKG=GasInv.PKG AND
Package.KProductID=@zKProductID),0)
                              END
                    IF @yNetbackTierLevel='COMPANY'
                              BEGIN
                                        SELECT @tmpVolumeTotal=ISNULL((SELECT
SUM(GasInv.PipelineActuals) FROM GasInv,Package
                                                  WHERE GasInv.GasMonth=@GasMonthx AND
GasInv.DBCR=0 AND GasInv.PriceType=1 AND GasInv.CID=@yCID
```

```
                                                        AND Package.PKG=GasInv.PKG AND
Package.KProductID=@zKProductID),0)
                                   END
          END
IF (@tmpVolumeTotal=0) OR (@tmpDaysInPeriod<1)
          BEGIN
                    SELECT @yAveragePerDay=0
          END
ELSE
          BEGIN
                    EXECUTE usp_GetProductVolumeRound @PIDx,@zRound OUTPUT
                    SELECT @yAveragePerDay=ROUND(@tmpVolumeTotal/@tmpDaysInPeriod,@zRound)
          END
/*
******************************************
* Determine which effective date of rules
* should be used.  This will be the max
* effective date where the effective date
* is either in or prior to the end of the
* current gas month. Only the set of rules
* associated with the most recent effective
* date will be used. If a date cannot be
* found then this function will return
* a zero percentage (ie. one isn't on
* the system that precedes the gas
* month).
******************************************
*/
SELECT @tmpMaxEffective=(SELECT MAX(effective) FROM knetback WHERE KID=(SELECT KID FROM package
WHERE PKG=@PIDx)
                              AND (effective<=@tmpEndDate) AND NetBackType=@TypeNetbackx)
IF @tmpMaxEffective IS NULL
          BEGIN
                    SELECT @tmpMaxEffective='01-01-1900'
          END
/*
******************************************
* If method 0 (all or nothing) then go
* and get the single tier percentage.
* The tier record will loop through and
* take the first tier record where the
* volume is greater than or equal then
* the average volume per day.
* This is the all or nothing netback
* pricing tier logic.
******************************************
*/
IF @yNetbackMethod=0
          BEGIN
                    SELECT @yDailyTotal=@yAveragePerDay
          END
ELSE
          BEGIN
                    SELECT @yDailyTotal=0
          END
/*
******************************************
* Initialize any fields that may be
* needed during the loop process.
******************************************
*/
SELECT @tmpAccumulatingTotal=@yAveragePerDay
SELECT @tmpPrevBand=0
SELECT @tmpAccumulatingPrice=0
/*
******************************************
* Now loop through all of the netback
* price records attached to the contract.
******************************************
*/
```

```
DECLARE NetbackCursor CURSOR LOCAL FORWARD_ONLY STATIC FOR
        SELECT
                effective,
                maxvollevel,
                netprice
                FROM
                        kNetBack
                WHERE
                        (KID=(SELECT KID FROM Package WHERE PKG=@PIDx)) AND
                        (effective=@tmpMaxEffective) AND
                        (maxvollevel>=@yDailyTotal) AND
                        (NetbackType=@TypeNetbackx)
                ORDER BY
                        maxvollevel asc
OPEN NetbackCursor
FETCH NEXT FROM NetbackCursor INTO @yEffective,@ymaxvollevel,@ynetprice
WHILE @@FETCH_STATUS = 0
        BEGIN
                IF @yNetbackMethod=0
                        BEGIN
                                IF @yNetbackPercentage=0
                                        BEGIN
                                                SELECT
@yNetbackPercentage=ROUND(@ynetprice,4)
                                        END
                        END
                /*
                ********************************************
                * If method 1 (accumulating) then go
                * through and weight each tier to derive
                * a percentage.  We know the total volume
                * for the month each tier will provide us
                * with the weighting information we need.
                ********************************************
                */
                IF @yNetbackMethod=1
                        BEGIN
                                IF @tmpAccumulatingTotal>0
                                        BEGIN
                                        SELECT @tmpCurrBand=(@ymaxvollevel-
@tmpPrevBand)
                                        IF @tmpCurrBand<=@tmpAccumulatingTotal
                                                BEGIN
                                                        SELECT
@tmpBandTotal=@tmpCurrBand
                                                        SELECT
@tmpAccumulatingTotal=(@tmpAccumulatingTotal-@tmpCurrBand)
                                                END
                                        ELSE
                                                BEGIN
                                                        SELECT
@tmpBandTotal=@tmpAccumulatingTotal
                                                        SELECT
@tmpAccumulatingTotal=0
                                                END
                                        SELECT @tmpBandWeightPerc=@tmpBandTotal
                                        SELECT
@tmpBandWeightPerc=@tmpBandWeightPerc/@yAveragePerDay
                                        SELECT
@tmpAccumulatingPrice=@tmpAccumulatingPrice+ROUND((@ynetprice*@tmpBandWeightPerc),4)
                                        END
                                SELECT @tmpPrevBand=@ymaxvollevel
                        END
                FETCH NEXT FROM NetbackCursor INTO @yEffective,@ymaxvollevel,@ynetprice
        END
CLOSE NetbackCursor
DEALLOCATE NetbackCursor
/*
********************************************
```

```
* Get the last accumulating price here
* and use this price...
*******************************************
*/
IF @yNetbackMethod=1
        BEGIN
                SELECT @yNetbackPercentage=@tmpAccumulatingPrice
        END

/*
*******************************************

* At this point a calculated netback
* percentage has been derived. Now
* check to see if the calculated netback
* percentage is less than the 'floor'
* amount setup on the contract.  If so,
* then use the floor amount.
*******************************************
*/
SELECT @ynetpricefloor=ISNULL((SELECT NetPriceFloor FROM K WHERE KID=(SELECT KID FROM Package
WHERE PKG=@PIDx)),0)
IF @ynetpricefloor<>0
        BEGIN
                IF @ynetpricefloor>@yNetbackPercentage
                        BEGIN
                                SELECT @yNetbackPercentage=@ynetpricefloor
                        END
        END
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  ON    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_fGetWASPIndicator(
                                        @PIDx INTEGER,
                                        @yWaspIndicator VARCHAR(10) OUTPUT
                                        )
AS
BEGIN
/*
*****************************************************************************************
Name: usp_fGetWaspIndicator

Description: This function will return the WASP indicator for the package ID that
was sent as an input parameter.  The WASP indicator is based on the combination of
deal classifications that have been established for this deal.  The default indicator
is 'None' (ie if classification information can't be found/etc.).  All combinations of
deal classifications should be setup.

Inputs: PIDx (package ID).

Outputs: A 'Comon' or 'Dedicated' or 'None' indicator which specifies whether
or not this package is considered 'WASP'able.

History:
```

```
05/12/1999 JAMIE Original Creation.

08/03/1999 JAMIE Modification to use the deal classification indicators
off of the package table versus the dealclass table.

*****************************************************************************************
*/
/*
********************************************
* Declare all exceptions, cursors
* and local variables that will be
* utilized by this procedure.
********************************************
*/
DECLARE @yDealContextID INTEGER
DECLARE @yDealTypeID INTEGER
DECLARE @yDealVolumeVolID INTEGER
DECLARE @yDealPricePeriodID INTEGER
DECLARE @yDealInterruptibleID INTEGER
/*
********************************************
* Populate the various deal classification
* identifiers based on the information
* stored on the package table.
********************************************
*/
SELECT
          @yDealContextID = PackageDBCR,
          @yDealTypeID = DealTypedcID,
          @yDealVolumeVolID = VolumeVolatilitydcID,
          @yDealPricePeriodID = PricePerioddcID,
          @yDealInterruptibleID = InterruptibledcID
          FROM
                    Package
          WHERE
                    PKG=@PIDx
/*
********************************************
* Now go and get the WASP indicator for
* this particular deal.
********************************************
*/
SELECT @yWaspIndicator=ISNULL((SELECT IncludeInWasp FROM rDealClassRules
                                        WHERE
                                                  DealContext=@yDealContextID AND
                                                  DealTypedcID=@yDealTypeID AND
                                                  VolumeVolatilitydcID=@yDealVolumeVolID AND
                                                  PricePerioddcID=@yDealPricePeriodID AND
                                                  InterruptibledcID=@yDealInterruptibleID),'None')
END


GO
SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_fGetWaspType(
                                                  @PIDx INTEGER,
                                                  @yWaspType VARCHAR(12) OUTPUT
                                                  )
AS
BEGIN
/*
*****************************************************************************************
Name: usp_fGetWaspType

Description: This function will return the WASP type field to use for the
```

```
specific package (deal) that is being looked at. This type is based on the
product id setup for the deal.

Inputs:

PIDx (package ID).

Outputs:

yWaspType - 'OIL','LIQUIDS', OR 'GAS'.

History:

12/03/2000  JAMIE  Original Creation.

************************************************************************************************
*/
/*
**********************************************
* Declare all exceptions, cursors
* and local variables that will be
* utilized by this procedure.
**********************************************
*/
DECLARE @yDealProduct VARCHAR(50)
DECLARE @yDealProductID INTEGER
/*
**********************************************
* Initialize the return value to be GAS
**********************************************
*/
SELECT @yWaspType='GAS'
/*
**********************************************
* Get the contrat ID off the deal
* (package) table.
**********************************************
*/
SELECT @yDealProductID = ISNULL((SELECT KProductID FROM package where PKG=@PIDx),0)
/*
**********************************************
* If a contract ID was found then
* based on the value then convert
* the netback type.
**********************************************
*/
IF @yDealProductID <> 0
        BEGIN
                SELECT @yDealProduct = ISNULL((SELECT shortdescription FROM SEProcessingCodes
WHERE processingcodeid= @yDealProductID),'Gas')
                IF @yDealProduct = 'Gas'
                        BEGIN
                                SELECT @yWaspType='GAS'
                        END
                IF @yDealProduct = 'Oil'
                        BEGIN
                                SELECT @yWaspType='OIL'
                        END
                IF @yDealProduct = 'Liquids'
                        BEGIN
                                SELECT @yWaspType='LIQUIDS'
                        END
        END
END
```

```
GO
SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON   SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_fIsLastDay(
                                                        @DT DATETIME
                                                        )
AS
BEGIN
DECLARE @LDx DATETIME
DECLARE @a INTEGER
EXECUTE usp_fLastDay @DT,@LDx OUTPUT
IF @LDx=@DT
        BEGIN
                SELECT @a=1
        END
ELSE
        BEGIN
                SELECT @a=0
        END
 RETURN(@a)
END


GO
SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON   SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_flastday(
                                        @lastdate DATETIME,
                                        @ldx DATETIME OUTPUT
                                        )
AS
BEGIN
/*
**************************************************
* Initially, just set the return value to be
* equal to the date coming in.
**************************************************
*/
SELECT @ldx=@lastdate
/*
**************************************************
* Now, loop thru adding 1 day to the date
* while the month is still equal.
**************************************************
*/
WHILE MONTH(@ldx)=MONTH(@lastdate)
        BEGIN
                SELECT @ldx=DATEADD(DAY,1,@ldx)
        END
/*
**************************************************
* Since the loop would have finished with
* the date being 1 day greater than the
* last day of the month, then back it off
* one day here to get the true end of
* month value...
**************************************************
*/
SELECT @ldx=DATEADD(DAY,-1,@ldx);
END
```

```
GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  ON    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_GasDayToGasMonth(
                                                  @GasDayX DATETIME,
                                                  @GasMonthX DATETIME OUTPUT
                                                  )
AS
BEGIN
/*
************************************************
* Initially, just set the return value to be
* equal to the date coming in.
************************************************
*/
SELECT @GasMonthX=@GasDayX
/*
************************************************
* Now, loop thru subtracting 1 day to the
* date while the month is still equal.
************************************************
*/
WHILE MONTH(@GasMonthX)=MONTH(@GasDayX)
        BEGIN
                SELECT @GasMonthX=DATEADD(DAY,-1,@GasMonthX)
        END
/*
************************************************
* Since the loop would have finished with
* the date being 1 day less than the
* first day of the month, then bump it up
* one day here to get the true beginning of
* month value...
************************************************
*/
SELECT @GasMonthX=DATEADD(DAY,1,@GasMonthX)
END


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  ON    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_GetProductVolumeRound(
                                    @PKGx INTEGER,
                                    @RoundNumber INTEGER OUTPUT
                                                  )
AS
/*
***********************************************************************************
Name: usp_GetProductRound

Description: Get the value used to round volumes to based on the information
in the processing codes table (typelimit field).

Inputs:

RoundNumber - Number of digits to round calculations too.

Outputs:
```

```
None

History:

11/23/2000 JAMIE Original creation.

***************************************************************************************
*/
BEGIN
DECLARE @zRoundNumber INTEGER
SELECT @zRoundNumber = ISNULL((SELECT SP.TypeLimit FROM SEProcessingCodes AS SP, Package WHERE
SP ProcessingCodeID = Package.KProductID AND Package.PKG=@PKGx),0);
SELECT @RoundNumber = @zRoundNumber
END




GO
SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_LinePrice(
                                                   @nETID INTEGER,
                                                   @nNomOrAct INTEGER
                                                   )
AS
BEGIN
/*
***************************************************************************************
Name: usp_LinePrice

Description: This procedure will calculate the line price for a specific Engine
record.  The input parameter nETID represents a unique key to a specific Engine
record.  In addition, the nNomOrAct parameter specifies whether or not to post the
price line information to the nomination area or the actual area of the engine
record. The volgroup field on the engine record contains the unique package (deal)
id.  This is used in the link to get the actual price components for the package.

Inputs:

nETID = Engine Key
nNomOrAct = (0=Nomination,1=Actualization)

Outputs:

Either an updated PriceOrRateNom or PriceOrRateAct field on the Engine record.
The precise field updated depends on the input parameter sent to this process (nNomOrAct).

History:

xx/xx/xx (?) CHIP Original Creation.

04/29/99 JAMIE Modified for WASP 2.10 Release. Structure changes
made to the Engine and Engine_Master tables. In
addition, all documentation added.  This particular
portion of the system required extensive changes
due to the need to store a nom and actual number
and because all price components are now stored
off the Engine_MasterPrice table (STID's 8 and 9).

06/22/2000 JAMIE Modified to pull in the entity, product and service
in order to get the correct price off the wasp table (values are passed
to the wasp routine).

11/10/2000 JAMIE Converted to Transact-SQL
```

```
**********************************************************************************
*/
/*
******************************************
* Declare all exceptions, cursors
* and local variables that will be
* utilized by this procedure.
******************************************
*/
DECLARE @xEngine_Effective DATETIME
DECLARE @xETID INTEGER
DECLARE @xSequenceNo INTEGER
DECLARE @xPriceTag VARCHAR(20)
DECLARE @xOperandVariable VARCHAR(1)
DECLARE @xPriceVariable VARCHAR(15)
DECLARE @xPriceEntryType VARCHAR(12)
DECLARE @xEffective DATETIME
DECLARE @xTID INTEGER
DECLARE @xEntityCID VARCHAR(12)
DECLARE @xKProductID INTEGER
DECLARE @xKServiceID INTEGER
DECLARE @yPrice DECIMAL(19,6)
DECLARE @yPriceInterimValue DECIMAL(19,6)
DECLARE @yMonthDate DATETIME
DECLARE @zTemp DECIMAL(19,6)

DECLARE Engine_MasterPriceAll CURSOR LOCAL FORWARD_ONLY STATIC FOR
                              SELECT DISTINCT
                                        emp.ETID,
                                        emp.SequenceNo,
                                        emp.PriceTag,
                                        emp.OperandVariable,
                                        emp.PriceVariable,
                                        pc.PriceEntryType,
                                        em.Effective,
                                        e.TID,
                                        k.entitycid,
                                        package.KProductID,
                                        package.KServiceID
                                        FROM
                                                  engine_masterprice AS emp,
                                                  engine AS e,
                                                  engine_master AS em,
                                                  pricecomponents AS pc,
                                                  gasinv,
                                                  k,
                                                  package
                                        WHERE
                                                  (e.ETID=@nETID) AND
                                                  (em.ETID=e.EM_ETID) AND
                                                  (emp.ETID=em.ETID) AND
                                                  (gasinv.tid=e.tid) AND
                                                  (k.kid=gasinv.kid) AND
                                                  (package.pkg=gasinv.pkg) AND
                                                  (pc.PriceTag=emp.PriceTag) AND
                                                  (emp.NomOrActual=@nNomOrAct)
                                        ORDER BY
                                                  emp.ETID,
                                                  emp.SequenceNo
/*
******************************************
* Initialize all fields here...
******************************************
*/
SELECT @yPrice=0
SELECT @yPriceInterimValue=0
/*
******************************************
* Open the cursor to get the pricing
* information and loop through all of
* the price component records. The end
* result price will ultimately be
* updated on the engine record.
*******************************************
*/
OPEN Engine_MasterPriceAll
FETCH NEXT FROM Engine_MasterPriceAll INTO
@xETID,@xSequenceNo,@xPriceTag,@xOperandVariable,@xPriceVariable,@xPriceEntryType,
                                    @xEffective,@xTID,@xEntityCID,@xKProductID,@xKServiceID
WHILE @@FETCH_STATUS = 0
          BEGIN
                    /*
                    *******************************************
                    * Derive the gas month based on the
                    * effective from the engine
                    * record.
                    *******************************************
                    */
                    SELECT @xEngine_Effective=(SELECT effective FROM engine WHERE ETID=@nETID)
                    EXECUTE usp_GasDayToGasMonth @xEngine_Effective,@yMonthDate OUTPUT
                    /*
                    *******************************************
                    * Convert the price variable portion to a
                    * number.  If an index then get the index
                    * amounts. The default price for any
                    * component not in this case statement is
                    * zero (ie.. WASP, UNKNOWN, etc.).
                    *******************************************
                    */
                    SELECT @yPriceInterimValue = 0
                    IF @xPriceEntryType='Numeric'
                              BEGIN
                                        SELECT @yPriceInterimValue=CAST(@xPriceVariable AS
DECIMAL(19,6))
                              END
                    IF @xPriceEntryType='Monthly IDX'
                              BEGIN
                                        EXECUTE usp_fGetIndex
@yMonthDate,@yMonthDate,@xPriceVariable,@yPriceInterimValue OUTPUT
                              END
                    IF @xPriceEntryType='Daily IDX'
                              BEGIN
                                        EXECUTE usp_fGetIndex
@yMonthDate,@xEngine_Effective,@xPriceVariable,@yPriceInterimValue OUTPUT
                              END
                    IF @xPriceEntryType='Basket IDX'
                              BEGIN
                                        EXECUTE usp_fGetIndexBasket
@yMonthDate,@xEngine_Effective,@xPriceVariable,@yPriceInterimValue OUTPUT
                              END
                    IF @xPriceEntryType='Wasp'
                              BEGIN
                                        EXECUTE usp_fGetCalcIndex
@xTID,@nNomOrAct,@xEntityCID,@xKProductID,@xKServiceID,@yMonthDate,@yPriceInterimValue OUTPUT
                              END
                    IF @yPriceInterimValue IS NULL
                              BEGIN
                                        SELECT @yPriceInterimValue = 0
                              END
                    /*
                    *******************************************
                    * At this point the yPriceInterim Value
                    * contains the individual price component
                    * amount.  Now, depending on the operator,
                    * apply this to the current total
                    * (yPrice). The end result is yPrice
                    * being updated with this component amount.
                    *******************************************
                    */
                    IF @xOperandVariable='+'
                            BEGIN
                                    SELECT @yPrice=@yPrice+@yPriceInterimValue
                            END
                    IF @xOperandVariable='-'
                            BEGIN
                                    SELECT @yPrice=@yPrice-@yPriceInterimValue
                            END
                    IF @xOperandVariable='*'
                            BEGIN
                                    SELECT @yPrice=@yPrice*@yPriceInterimValue
                            END
                    IF @xOperandVariable='/'
                            BEGIN
                                    IF (@yPrice<>0) AND (@yPriceInterimValue<>0)
                                            BEGIN
                                                    SELECT
@yPrice=@yPrice/@yPriceInterimValue
                                            END
                            END
                    FETCH NEXT FROM Engine_MasterPriceAll INTO
@xETID,@xSequenceNo,@xPriceTag,@xOperandVariable,@xPriceVariable,@xPriceEntryType,

        @xEffective,@xTID,@xEntityCID,@xKProductID,@xKServiceID
        END
CLOSE Engine_MasterPriceAll
DEALLOCATE Engine_MasterPriceAll
/*
*********************************************
* Finally, take the results of the price
* components and update the engine record
* with the price.
*
* If nom calculation then update then
* nom price.  If actual calculation then
* update the actualized price.
*********************************************
*/
IF @nNomOrAct=0
        BEGIN
                UPDATE
                        Engine
                        SET
                                PriceOrRateNom=@yPrice
                        WHERE
                                ETID=@nETID
        END
IF @nNomOrAct=1
        BEGIN
                UPDATE
                        Engine
                        SET
                                PriceOrRateAct=@yPrice
                        WHERE
                                ETID=@nETID
        END
END



GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_message(
                                        @messagex VARCHAR(255)
                                        )
AS
/*
*************************************************************************************
Name: usp_message

Description: This routine will dictate where and how messages from the system
will (or will not) be posted.  These are transitory messages generated by the
system (like during a calculation).

Inputs:

messagex - Text message to write

Outputs:

None


History:

11/07/2000 JAMIE Original creation.

*************************************************************************************
*/
BEGIN
DECLARE @tmpMessage VARCHAR(254)
/*
INSERT INTO ApplicationMessages (ApplicationMessageText) VALUES (@messagex)

PRINT @messagex
*/

SELECT @tmpMessage = @messagex
END










GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPrice(
                                        @PIDx INTEGER,
                                        @WhichPricex INTEGER,
                                        @GasMonthx DATETIME,
                                        @DBCRx INTEGER
                                        )
AS
BEGIN
/*
*************************************************************************************
Name: usp_PSPrice
```

```
Description: Price all of the gas inventory items.

History:

xx/xx/xx (?) CHIP Original Creation.

05/03/99 JAMIE Modified for WASP 2.10 Release. Structure changes
made to the Engine and Engine_Master tables. In
addition, all documentation added. In addition modifications were made to drive the
pricing off package identifier versus Gas Inventory Transaction Identifier (TID). Since
all pricing is done at a package level.

Only those entries within the gas inventory with pricetype=1
will be processed by this procedure. These entries represent
only the purchase and sale items AND SHOULD HAVE Engine_Master
records associated with them.

07/12/2000 JAMIE Modified to check for the actualizedflag on the
gasinv record. If the flag is set to a 'Y' then set the price accordingly. If
the flag is set to something other than a 'Y' (ie.. 'N' or null) then the
price will automatically get a zero. The price or rate number for actuals
will still calculate AND it is possible that some meters within a deal will
calculate (if the flag is set) while other meters on the same deal will not
(if the flag is not set). The engine record is where all calculated results
are stored and will contain zeros for the entries that have not been
setup to be actualized.

******************************************************************************************
*/
/*
*********************************************
* Declare all variables and cursors
* that are needed by this process.
*********************************************
*/
DECLARE @tmpEndDate DATETIME
DECLARE @tmpNextEffectiveDate DATETIME
DECLARE @tmpNumberDays INTEGER
DECLARE @tmpVolumeInPeriod DECIMAL(19,2)
DECLARE @tmpDateToUse DATETIME
DECLARE @yTID INTEGER
DECLARE @yActualizedFlag VARCHAR(1)
DECLARE @ySTID INTEGER
DECLARE @yEffective DATETIME
DECLARE @yETID INTEGER

DECLARE @zRound INTEGER

DECLARE GasInventoryCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    DISTINCT
                    TID,
                    ActualizedFlag
                    FROM
                              GasInv
                    WHERE
                              (PKG=@PIDx) AND
                              (PriceType=1) AND
                              (DBCR=@DBCRx)
/*
*********************************************
* At this point the calculation needs to
* happen. Iterate through each of the
* inventory items attached to this particular
* package... Only STID's of 8 or 9 are
* priced here... (STID=8 is DBCR=0 is a
* purchase),(STID=9 is DBCR=1 is a sale).
*
* Within each inventory item go through
* each effective date/STID and use the
* pricing rules to determine whether the
* pricing accumulates or is all or
* nothing.
*******************************************
*/
EXECUTE usp_GetProductVolumeRound @PIDx,@zRound OUTPUT
OPEN GasInventoryCursor
FETCH NEXT FROM GasInventoryCursor INTO @yTID,@yActualizedFlag
WHILE @@FETCH_STATUS = 0
        BEGIN
                DECLARE EngineCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                        SELECT
                                DISTINCT
                                e.ETID,
                                e.Effective,
                                e.STID,
                                e.TID
                                FROM
                                        Engine AS e,
                                        Engine_Master AS em
                                WHERE
                                        (em.ETID=e.EM_ETID) AND
                                        (em.PID=e.VolGroup) AND
                                        (e.TID=@yTID)
                                ORDER BY
                                        e.ETID
                OPEN EngineCursor
                FETCH NEXT FROM EngineCursor INTO @yETID,@yEffective,@ySTID,@yTID
                WHILE @@FETCH_STATUS = 0
                        BEGIN
                                /*
                                *******************************************
                                * Calculate and update the engine with the
                                * the actual price from the engine_master
                                * via call to the following function.
                                *******************************************
                                */
                                EXECUTE usp_LinePrice @yETID,@WhichPricex
                                /*
                                *******************************************
                                * Determine the volume total to be applied
                                * to this price line here. This represents
                                * the sum of the volume between the
                                * effective date and the end of the
                                * month OR the next price effective
                                * date for this item.  The value of
                                * tmpNumberDays contains the number of
                                * days to apply the price and volumes
                                * toward within the calculation.
                                *******************************************
                                */
                                EXECUTE usp_fLastDay @GasMonthx,@tmpEndDate OUTPUT
                                SELECT @tmpNextEffectiveDate=(SELECT MIN(effective)-1 FROM
engine AS e WHERE (e.TID=@yTID) AND (e.STID=@ySTID) AND (e.Effective>@yEffective))
                                IF @tmpNextEffectiveDate IS NULL
                                        BEGIN
                                                SELECT
@tmpNextEffectiveDate=@tmpEndDate
                                        END
                                IF @tmpNextEffectiveDate<@tmpEndDate
                                        BEGIN
                                                SELECT
@tmpDateToUse=@tmpNextEffectiveDate
                                        END
                                ELSE
                                        BEGIN
                                                SELECT @tmpDateToUse=@tmpEndDate
                                        END
                                        SELECT
@tmpNumberDays=DATEDIFF(day,@yEffective,@tmpDateToUse) + 1
                                        IF @WhichPricex=0
                                              BEGIN
                                                    SELECT
@tmpVolumeInPenod=ISNULL((SELECT SUM(Nom) FROM GasInvD WHERE (GasInvD.TID=@yTID)

      AND (GasInvD.GasDay BETWEEN @yEffective AND @tmpDateToUse)),0)
                                              END
                                        IF @WhichPricex=1
                                              BEGIN
                                                    IF @yActualizedFlag='Y'
                                                          BEGIN
                                                          SELECT
@tmpVolumeInPeriod=ISNULL((SELECT SUM(PipelineActuals) FROM GasInvD WHERE (GasInvD.TID=@yTID)

                  AND (GasInvD.GasDay BETWEEN @yEffective AND @tmpDateToUse)),0)
                                                          END
                                                    ELSE
                                                          BEGIN
                                                          SELECT @tmpVolumeInPeriod=0
                                                          END
                                              END
                                        /*
                                        ******************************************
                                        * Update the actual engine volumes and
                                        * amounts here...
                                        ******************************************
                                        */
                                        IF @WhichPricex=0
                                              BEGIN
                                                    UPDATE
                                                          Engine
                                                          SET

      Volume=ROUND(@tmpVolumeInPeriod,@zRound),

      Amount=ROUND((@tmpVolumeInPeriod*Engine.PriceOrRateNom),2)
                                                    WHERE
                                                                ETID=@yETID
                                              END
                                        IF @WhichPricex=1
                                              BEGIN
                                                    UPDATE
                                                          Engine
                                                          SET

      VolumeAct=ROUND(@tmpVolumeInPeriod,@zRound),

      AmountAct=ROUND((@tmpVolumeInPeriod*Engine.PriceOrRateAct),2)
                                                    WHERE
                                                                ETID=@yETID
                                              END
                                        FETCH NEXT FROM EngineCursor INTO
@yETID,@yEffectve,@ySTID,@yTID
                                  END
                  CLOSE EngineCursor
                  DEALLOCATE EngineCursor
                  FETCH NEXT FROM GasInventoryCursor INTO @yTID,@yActualizedFlag
      END
CLOSE GasInventoryCursor
DEALLOCATE GasInventoryCursor
END


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO
```

```
SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_PSPriceAll(
                                                    @GasMonthx DATETIME,
                                                    @DebitCreditx INTEGER,
                                                    @WhichPricex INTEGER,
                                                    @PKGx INTEGER,
                                                    @EntityCIDx VARCHAR(12),
                                                    @IncludeInWASPx VARCHAR(10)
                                                    )
AS
BEGIN
/*
***********************************************************************************
Name: usp_PSPriceAll

Description:

Loop thruough all packages (deals) involved within a given month (purchase
or sale) and invoke the price procedures.

Inputs:

GasMonthx (Gas Month to price),
DebitCreditx (0=Debit (Purchases) - 1=Credit (Sales)),
WhichPricex (0=Nominations, 1=Actualizations
PKGx (0=all otherwise specific package ID)
EntityCIDx (owning company entity id)
IncludeInWASPx ('' for all, otherwise check for 'Common','Dedicated', or 'None')

History:

05/13/99 JAMIE This entire process was rewritten with V2.10 of
the Gas Control System.  Package driven now
instead of individual inventory item driven.

07/22/99 JAMIE Include 3rd party deals within the
calcualtion process.  They WILL NOT BE included within the WASP calculations
and will be treated the same as "Dedicated" (sanctioned sales) deals.  This
will ensure they are not affecting any other pricing component.

05/24/2000 JAMIE Modified to include the changes to calculate based on company
entity ID (passed to this calculation).  This ensures that WASP calculations/etc
are all within their respective companies...  The deal cursor (PackageCursor)
will now only select those items where the entity ID for the contract on the deal
matches the one passed to this routine.

07/26/2000 JAMIE Modified to include the IncludeInWaspx parameter to
this particular procedure.  This will allow certain types of deals to
be priced independently of other types (ie.. do 3rd party first in order
to divie the proceeds either to a pool OR to another deal).

***********************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zTypeText VARCHAR(10)
DECLARE @zMessage VARCHAR(255)

DECLARE @yPKG INTEGER
DECLARE @yIncludeInWasp VARCHAR(10)

DECLARE PackageCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
```

```
                    DISTINCT
                    GasInv.PKG
                    FROM
                              GasInv,
                              Package,
                              K
                    WHERE
                              Package.PKG=GasInv.PKG AND
                              K.KID=Package.KID AND
                              K.EntityCID=@EntityCIDx AND
                              GasInv.GasMonth=@GasMonthx AND
                              GasInv.DBCR=@DebitCreditx AND
                              GasInv.PriceType=1
                    ORDER BY
                              GasInv.PKG
/*
*******************************************
* Initialize any fields required.
*******************************************
*/
IF @DebitCreditx=0
          BEGIN
                    SELECT @zTypeText='Purchase'
          END
IF @DebitCreditx=1
          BEGIN
                    SELECT @zTypeText='Sale'
          END
/*
*******************************************
* Loop through each package
* involved in this calculation.  As each
* deal is fetched get its WASP indicator
* information in order to determine if
* it can be involved in this process.
*******************************************
*/
OPEN PackageCursor
FETCH NEXT FROM PackageCursor INTO @yPKG
WHILE @@FETCH_STATUS = 0
          BEGIN
                    BEGIN TRANSACTION
                    EXECUTE usp_fGetWASPIndicator @yPKG,@yIncludeInWasp OUTPUT
                    IF (@PKGx=0) OR ((@PKGx<>0) AND (@PKGx=@yPKG))
                              BEGIN
                                        IF (@IncludeInWaspx='') OR (@IncludeInWaspx=@yIncludeInWasp)
                                                  BEGIN
                                                            SELECT @zMessage =  'PSPriceAll Running
for Entity '+@EntityCIDx+' and type '+@zTypeText+', Package:'+'  '+CONVERT(VARCHAR(10),@yPKG)+'....'
                                                            EXECUTE usp_message @zMessage
                                                            /*
                                                            *******************************************
                                                            * Reset the financial override dollar
                                                            * amount to zeros at the beginning of the
                                                            * calculate for the deal...

                                                            *******************************************
                                                            */
                                                            IF @WhichPricex=0
                                                                      BEGIN
                                                                      UPDATE
                                                                                          package
                                                                                          SET
          FinancialNomAmount=0
                                                                                          WHERE

          PKG=@yPKG
                                                                      END
```

```
IF @WhichPricex=1
        BEGIN
        UPDATE
                package
                SET
FinancialActAmount=0
                WHERE
PKG=@yPKG
        END
/*
**********************************************
* Create any system generated pricing
* components for this package...  These
* pricing components are tightly related
* to the Engine_Master.  This is needed
* to be done prior to populating the
* Engine with pricing information.
**********************************************
*/
EXECUTE usp_PSPriceComponentsCheck
@yPKG,@WhichPricex,@GasMonthx,@DebitCreditx
/*
**********************************************
* Now create all engine items for all
* Engine_Master records.  This is where
* the engine will be populated with entries
* based on information stored in the
* Engine_Master.  Included is the daily
* index price 'proliferation' routine.
**********************************************
*/
EXECUTE usp_PSPricePopulateEngine
@yPKG,@WhichPricex,@GasMonthx,@DebitCreditx
/*
**********************************************
* Finally, now go and price the actual
* engine entries that were created in the
* previous step...
**********************************************
*/
EXECUTE usp_PSPrice
@yPKG,@WhichPricex,@GasMonthx,@DebitCreditx
                        END
                END
            COMMIT WORK
            FETCH NEXT FROM PackageCursor INTO @yPKG
        END
CLOSE PackageCursor
DEALLOCATE PackageCursor
END

GO
SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceAnyNewInvoicesNeeded(
                        @GasMonthx DATETIME,
                        @EntityCIDx VARCHAR(12)
                        )
```

```
AS
BEGIN
/*
******************************************************************************************
Name: usp_PSPriceAnyNewInvoicesNeeded

Description:

This routine gets executed once a gas month has been put
in an 'Invoiced' status.  It will automatically go out and assign and invoice
number where one previously did not exist (could use the same invoice
number as an existing).

Inputs:

GasMonthx - Gas month being calculated
EntityCIDx - owning company

History:

12/15/1999 JAMIE   Original creation

12/21/1999 JAMIE Modify to put the monthly aphabetic code as the first
field of the invoice number to eliminate OGSYS clipping of a leading zero.

05/24/2000 JAMIE Modified to only create the invoices within the given owning
company.  The invoice numbers will need to be unique within the entire system.

******************************************************************************************
*/
/*
**********************************************
* Declare all variables and cursors
* that are needed by this process.
**********************************************
*/
DECLARE @yTID INTEGER
DECLARE @yCID VARCHAR(12)
DECLARE @yPipe VARCHAR(12)
DECLARE @zAcctgIdentifier VARCHAR(12)
DECLARE @zYear INTEGER
DECLARE @zYearString VARCHAR(1)
DECLARE @zMonth INTEGER
DECLARE @zMonthString VARCHAR(1)
DECLARE @zNumToUse INTEGER
DECLARE @zNumToUseLength INTEGER
DECLARE @zNumToUseString VARCHAR(3)
DECLARE @zNumToUseZeros VARCHAR(3)
DECLARE @zMaxAcctgIdentifier VARCHAR(12)
DECLARE @zWorkString VARCHAR(12)

DECLARE GasInvCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                GasInv.TID,
                GasInv.CID,
                GasInv.PipeField
                FROM
                        GasInv,
                        Package,
                        K
                WHERE
                        GasInv.GasMonth=@GasMonthx AND
                        GasInv.PriceType=1 AND
                        GasInv.DBCR=1 AND
                        (AcctgIdentifier IS NULL OR AcctgIdentifier='') AND
                        Package.PKG=GasInv.PKG AND
                        K.KID=Package.KID AND
                        K.EntityCID=@EntityCIDx
                ORDER BY
                        GasInv.CID,
/*
*********************************************
* Determine the prefix to use for the
* creation of the invoice numbers.  If more
* than 10 years then these numbers begin
* to be reused.
*
* This routine is CHEAP but it should
* suffice.
*********************************************
*/
SELECT @zYear=YEAR(@GasMonthx)
SELECT @zYearString=RIGHT(CONVERT(VARCHAR(4),@zYear),1)
SELECT @zMonth=MONTH(@GasMonthx)
IF @zMonth=1
          BEGIN
                    SELECT @zMonthString='A'
          END
IF @zMonth=2
          BEGIN
                    SELECT @zMonthString='B'
          END
IF @zMonth=3
          BEGIN
                    SELECT @zMonthString='C'
          END
IF @zMonth=4
          BEGIN
                    SELECT @zMonthString='D'
          END
IF @zMonth=5
          BEGIN
                    SELECT @zMonthString='E'
          END
IF @zMonth=6
          BEGIN
                    SELECT @zMonthString='F'
          END
IF @zMonth=7
          BEGIN
                    SELECT @zMonthString='G'
          END
IF @zMonth=8
          BEGIN
                    SELECT @zMonthString='H'
          END
IF @zMonth=9
          BEGIN
                    SELECT @zMonthString='I'
          END
IF @zMonth=10
          BEGIN
                    SELECT @zMonthString='J'
          END
IF @zMonth=11
          BEGIN
                    SELECT @zMonthString='K'
          END
IF @zMonth=12
          BEGIN
                    SELECT @zMonthString='L'
          END
/*
*********************************************
* Find the starting point to begin
* assigning new invoices from just
* in case some numbers need to be
* assigned.
*********************************************
```

09855014.052801

```
*/
SELECT @zNumToUse=0
SELECT @zMaxAcctgIdentifier=(SELECT max(AcctgIdentifier) FROM GasInv WHERE GasMonth=@GasMonthx AND
DBCR=1 AND PriceType=1)
IF LEN(@zMaxAcctgIdentifier) = 6
        BEGIN
                SELECT @zWorkString=RIGHT(@zMaxAcctgIdentifier,4)
                SELECT @zWorkString=LEFT(@zWorkString,3)
                SELECT @zNumToUse=CONVERT(INTEGER,@zWorkString)
        END
/*
*******************************************
* Now go get the records that do not
* yet have a invoice number assigned
* to them (ie. execute the cursor).
*******************************************
*/
OPEN GasInvCursor
FETCH NEXT FROM GasInvCursor INTO @yTID,@yCID,@yPipe
WHILE @@FETCH_STATUS = 0
        BEGIN
                /*
                *******************************************
                * Now go and find one, if one exists.
                *******************************************
                */
                SELECT @zAcctgIdentifier=(SELECT DISTINCT(AcctgIdentifier) FROM GasInv WHERE
GasMonth=@GasMonthx AND
                                                                DBCR=1 AND PriceType=1 AND CID=@yCID
AND PipeField=@yPipe AND AcctgIdentifier IS NOT NULL AND AcctgIdentifier<>'')

                IF @zAcctgIdentifier IS NULL
                        BEGIN
                                /*
                                *******************************************
                                * For each of these combinations generate
                                * and invoice number and update the GasInv
                                * table... Make sure that the number
                                * to use is padded with zeros in order
                                * to create a complete invoice number.
                                * REALLY CHEAP ZERO PADDING.
                                *******************************************
                                */
                                SELECT @zNumToUse=@zNumToUse+1
                                SELECT
@zNumToUseString=CONVERT(VARCHAR(3),@zNumToUse)
                                SELECT @zNumToUseLength=LEN(@zNumToUseString)
                                SELECT @zNumToUseZeros=''
                                IF @zNumToUseLength < 3
                                        BEGIN
                                        IF @zNumToUseLength=2
                                                        BEGIN
                                                        SELECT @zNumToUseZeros='0'
                                                        END
                                        IF @zNumToUseLength=1
                                                        BEGIN
                                                        SELECT
@zNumToUseZeros='00'
                                                        END
                                        END;
                                SELECT
@zAcctgIdentifier=@zMonthString+@zYearString+@zNumToUseZeros+@zNumToUseString+'N'
                                /*
                                *******************************************
                                * Finally, post the invoice number that
                                * was just created to the gas inventory
                                * table.
                                *******************************************
                                */
                                UPDATE
```

```
                                                GasInv
                                                SET
                                                        AcctgIdentifier=@zAcctgIdentifier
                                                WHERE
                                                        GasMonth=@GasMonthx AND
                                                        DBCR=1 AND
                                                        PriceType=1 AND
                                                        CID=@yCID AND
                                                        PipeField=@yPipe AND
                                                        TID=@yTID
                                END
                        FETCH NEXT FROM GasInvCursor INTO @yTID,@yCID,@yPipe
                END
CLOSE GasInvCursor
DEALLOCATE GasInvCursor
END



GO
SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_PSPriceAssignInvoiceNo(
                                                        @GasMonthx DATETIME
                                                        )
AS
BEGIN
SET NOCOUNT ON
/*
*****************************************************************************************
Name: usp_PSPriceAssignInvoiceNo

Description: This routine will clear out any existing invoice numbers on the gas
inventory table AND generate/assign an invoice number and post to the gas
inventory table.

This particular routine is only looking at 'Sales' (DBCR=1) within the specified
gas month (GasMonthx) that have a price type of '1' (ie.. not a transport inventory
item).

The format of the invoice number that gets generated will be as follows:

Character
---------
1  Represents alph code for month (A=January, B=February, etc.).
2  Represents the last digit of the year (1999=9, 2000=0, etc.).
3-5 Represents unqiue number assigned.
6  Represents 'N' for Nominations.

These invoice numbers are generated uniquely for all sales meters within a given pipe and
company identifier.  This procedure will assign the invoice number to both the
nom and actual fields.  Later (during actual calculations) the actual invoice number may
or may not get updated based on the modifications made to the volumes or prices.

Inputs:  GasMonthx (Gas Month to calculate),

History:

10/27/1999 JAMIE   Original creation

11/19/1999 JAMIE  Modified the number creation to post the final character as
an 'N'.

12/21/1999 JAMIE  Modified the number creation process to put the monthly
alphabetic code at the beginning of the invoice number instead of the 2nd
```

character.

```
******************************************************************************************
*/
/*
******************************************
* Declare all variables and cursors
* that are needed by this process.
******************************************
*/
DECLARE @yCID VARCHAR(12)
DECLARE @yPipe VARCHAR(12)
DECLARE @zAcctgIdentifier VARCHAR(12)
DECLARE @zYear INTEGER
DECLARE @zYearString VARCHAR(1)
DECLARE @zMonth INTEGER
DECLARE @zMonthString VARCHAR(1)
DECLARE @zNumToUse INTEGER
DECLARE @zNumToUseLength INTEGER
DECLARE @zNumToUseString VARCHAR(3)
DECLARE @zNumToUseZeros VARCHAR(3)
/*
******************************************
* Determine the prefix to use for the
* creation of the invoice numbers.  If more
* than 10 years then these numbers begin
* to be reused.
*
* This routine is CHEAP but it should
* suffice.
******************************************
*/
SELECT @zYear=YEAR(@GasMonthx)
SELECT @zYearString=RIGHT(CONVERT(VARCHAR(4),@zYear),1)
SELECT @zMonth=MONTH(@GasMonthx)
IF @zMonth=1
        BEGIN
                SELECT @zMonthString='A'
        END
IF @zMonth=2
        BEGIN
                SELECT @zMonthString='B'
        END
IF @zMonth=3
        BEGIN
                SELECT @zMonthString='C'
        END
IF @zMonth=4
        BEGIN
                SELECT @zMonthString='D'
        END
IF @zMonth=5
        BEGIN
                SELECT @zMonthString='E'
        END
IF @zMonth=6
        BEGIN
                SELECT @zMonthString='F'
        END
IF @zMonth=7
        BEGIN
                SELECT @zMonthString='G'
        END
IF @zMonth=8
        BEGIN
                SELECT @zMonthString='H'
        END
IF @zMonth=9
        BEGIN
                SELECT @zMonthString='I'
```

```
		END
	IF @zMonth=10
		BEGIN
			SELECT @zMonthString='J'
		END
	IF @zMonth=11
		BEGIN
			SELECT @zMonthString='K'
		END
	IF @zMonth=12
		BEGIN
			SELECT @zMonthString='L'
		END
	/*
	*******************************************
	* Clear out the invoice number that may
	* have preexisted for this particular
	* gas month (this number will always be
	* empty UNLESS the gas month is opened
	* and closed more than once).
	*******************************************
	*/
	BEGIN TRANSACTION
	UPDATE
		GasInv
		SET
			AcctgIdentifier=NULL
		WHERE
			GasMonth=@GasMonthx AND
			DBCR=1 AND
			PriceType=1 AND
			(AcctgIdentifier IS NOT NULL OR AcctgIdentifier<>'')
	COMMIT WORK
	/*
	*******************************************
	* Now build a cursor that contains all of
	* the unique combinations of company and
	* pipeline (ordered by company and pipeline).
	*******************************************
	*/
	SELECT @zNumToUse=0
	DECLARE GasInvCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
		SELECT
			DISTINCT
			(GasInv.CID),
			(GasInv.PipeField)
			FROM
				GasInv
			WHERE
				GasInv.GasMonth=@GasMonthx AND
				GasInv.PriceType=1 AND
				GasInv.DBCR=1
			ORDER BY
				GasInv.CID,
				GasInv.PipeField
	OPEN GasInvCursor
	FETCH NEXT FROM GasInvCursor INTO @yCID,@yPipe
	WHILE @@FETCH_STATUS = 0
		BEGIN
			BEGIN TRANSACTION
			/*
			*******************************************
			* For each of these combinations generate
			* and invoice number and update the GasInv
			* table... Make sure that the number
			* to use is padded with zeros in order
			* to create a complete invoice number.
			* REALLY CHEAP ZERO PADDING.
			*******************************************
			*/
```

```
                    SELECT @zNumToUse=@zNumToUse+1
                    SELECT @zNumToUseString=CONVERT(VARCHAR(3),@zNumToUse)
                    SELECT @zNumToUseLength=LEN(@zNumToUseString)
                    SELECT @zNumToUseZeros=''
                    IF @zNumToUseLength < 3
                            BEGIN
                                    IF @zNumToUseLength=2
                                            BEGIN
                                                    SELECT @zNumToUseZeros='0'
                                            END
                                    IF @zNumToUseLength=1
                                            BEGIN
                                                    SELECT @zNumToUseZeros='00'
                                            END
                            END
                    SELECT
@zAcctgIdentifier=@zMonthString+@zYearString+@zNumToUseZeros+@zNumToUseString+'N'
                    /*
                    ********************************************
                    * Finally, post the invoice number that
                    * was just created to the gas inventory
                    * table.
                    ********************************************
                    */
                    UPDATE
                            GasInv
                            SET
                                    AcctgIdentifier=@zAcctgIdentifier
                            WHERE
                                    GasMonth=@GasMonthx AND
                                    DBCR=1 AND
                                    PriceType=1 AND
                                    CID=@yCID AND
                                    PipeField=@yPipe
                    COMMIT WORK
                    FETCH NEXT FROM GasInvCursor INTO @yCID,@yPipe
            END
CLOSE GasInvCursor
DEALLOCATE GasInvCursor
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceAuto
AS
BEGIN
/*
***********************************************************************************************
Name: usp_PSPriceAuto

Description:

This procedure will be scheduled at automatically calculate the gas months
in their respective stages.  Noms get calculated for gas months in the 'Sales' stage.
Pipeline actuals get calculated for gas months in the 'Invoiced' stage.  All other gas
months are ignored by this process.

Inputs:

None
```

```
History:

07/29/1999 JAMIE Original Creation.

10/20/1999 JAMIE Modified to invoke the PSPriceCostAll routine which will
calculate other costs for deals and post them to the engine table.

03/22/2000 JAMIE Modified to invoke the single month calculation routine. This will
ensure easier (non duplicated) maintenance on procedures to update price calculations.

****************************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @yGasMonth DATETIME
/*
*******************************************
* First, calculate all of the nom
* numbers (each gas month).
*******************************************
*/
DECLARE GasMonthCursor1 CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    GasMonth
                    FROM
                              rGasMonth
                    WHERE
                              CurrentStatus='Sales' AND
                              (LockedUser IS NULL OR LockedUser='')
                    ORDER BY
                              GasMonth
OPEN GasMonthCursor1
FETCH NEXT FROM GasMonthCursor1 INTO @yGasMonth
WHILE @@FETCH_STATUS = 0
          BEGIN
                    EXECUTE usp_PSPriceAutoMonth @yGasMonth,0
                    FETCH NEXT FROM GasMonthCursor1 INTO @yGasMonth
          END
CLOSE GasMonthCursor1
DEALLOCATE GasMonthCursor1
/*
*******************************************
* Now calculate based on the pipeline
* actuals each month.
*******************************************
*/
DECLARE GasMonthCursor2 CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    GasMonth
                    FROM
                              rGasMonth
                    WHERE
                              CurrentStatus='Invoiced' AND
                              (LockedUser IS NULL OR LockedUser='')
                    ORDER BY
                              GasMonth
OPEN GasMonthCursor2
FETCH NEXT FROM GasMonthCursor2 INTO @yGasMonth
WHILE @@FETCH_STATUS = 0
          BEGIN
                    EXECUTE usp_PSPriceAutoMonth @yGasMonth,1
                    FETCH NEXT FROM GasMonthCursor2 INTO @yGasMonth
          END
CLOSE GasMonthCursor2
DEALLOCATE GasMonthCursor2
```

```
END


GO
SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceAutoMonth(
                                                        @GasMonthx DATETIME,
                                                        @WhichVolumex INTEGER
                                                                       )
AS
BEGIN
SET NOCOUNT ON
/*
******************************************************************************************
Name: usp_PSPriceAutoMonth

Description:

This procedure will be execute all of the price calculation procedures
required for a given month INCLUDING locking the month from other executions... This
particualr procedure will be executed asynchronously by the system through the online
screens.

Inputs:

GasMonthx (Gas month to calculate)
WhichVolumex (Price noms=0, Price actuals=1)

History:

08/31/1999 JAMIE Original Creation.

12/15/1999 JAMIE Modified to execute a new stored procedure once
the gas month has been changed to the 'Accounting' status.  This new
procedure will mark and 'zap' the invoice numbers (amongst other things)
on those gas inventory items were some sort of a price or volume adjustment
was made.

03/22/2000 JAMIE Modified this process to handle all of the calculations for
gas months, etc. Moved the 'Divie' process to this routine (was buried within
the transport cost module).

05/24/2000 JAMIE Modified to enable an outer cursor on company entity (CID).  This will
allow for the partitioning of the calculations based on company ID (so we don't mix
WASP Pool results/etc.).

07/26/2000 JAMIE Modified to incorporate the changes to process calculations for certain
types of deals prior to others (ie. 3rd party first so that profits can be distributed.
This change included passing a new parameter to the PSPriceAll function (on which
pool ('' for all)...

08/25/2000 JAMIE Modified to remove logic that invoked the older calculation
routines.

02/01/2001 JAMIE Modified to remove the transport section (commented out).

******************************************************************************************
*/
DECLARE @yCIDEntity VARCHAR(12)
DECLARE @yGasMonth DATETIME
DECLARE @yCurrentStatus VARCHAR(20)

DECLARE EntityCIDCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
```

```
                SELECT
                        CID
                        FROM
                                company
                        WHERE
                                EntityIndicator='Y'
                        ORDER BY
                                CID
        /*
        *******************************************
        * Execute a cursor to calculate based on
        * entity.
        *******************************************
        */
        OPEN EntityCIDCursor
        FETCH NEXT FROM EntityCIDCursor INTO @yCIDEntity
        WHILE @@FETCH_STATUS = 0
                BEGIN
                        /*
                        *******************************************
                        * Now only calculate if the month
                        * is not currently involved with a
                        * calculation of some sort (month
                        * needs to be unlocked).
                        *
                        * If the status was modified and the
                        * current status in 'Invoiced' then
                        * go and build all of the pipeline
                        * actuals.
                        *******************************************
                        */
                        DECLARE GasMonthCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                                SELECT
                                        GasMonth,
                                        CurrentStatus
                                        FROM
                                                rGasMonth
                                        WHERE
                                                GasMonth=@GasMonthx AND
                                                (LockedUser IS NULL OR LockedUser='')
                        OPEN GasMonthCursor
                        FETCH NEXT FROM GasMonthCursor INTO @yGasMonth,@yCurrentStatus
                        WHILE @@FETCH_STATUS = 0
                                BEGIN
                                        /*
                                        *******************************************
                                        * Indicate that the gas month is in
                                        * progress so that no one else tryes to
                                        * calculate at the same time.
                                        *******************************************
                                        */
                                        UPDATE
                                                rGasMonth
                                                SET
                                                        LockedUser='PSPriceAutoM',
                                                        LockedDate=getdate()
                                                WHERE
                                                        GasMonth=@GasMonthx
                                        /*
                                        *******************************************
                                        * Calculate prices on all sales deals...
                                        *******************************************
                                        */
                                        EXECUTE usp_PSPriceAll
        @GasMonthx,1,@WhichVolumex,0,@yCIDEntity,''
                                        /*
                                        *******************************************
                                        * Calculate 'Other Costs' associated to
                                        * all sales deals (required here in
                                        * order to post the other cost amounts
```

```
* to WASP pools/etc...
*********************************************
*/
EXECUTE usp_PSPriceCostAll
@GasMonthx,@WhichVolumex,@yCIDEntity,1,''
/*
*********************************************
* Now create the temporary WASPRouting
* table entries for all products, services
* and wasp types.  The calculations will
* not 'walk back' from sale to purchase
* here (unless OLD routing month)...
*********************************************
*/
EXECUTE usp_PSPriceWASPCalc
@GasMonthx,@WhichVolumex,@yCIDEntity
/*
*********************************************
* If new routing method then resolve based
* on entity and IncludeInWasp pool. This
* is done this way in order to potentially
* distribute proceeds from 3rd party
* deals back to either a WASP pool meter
* or to another deal...
*
* 1. Resolve and price 'None' pool.
* 2. Divie out any proceeds.
* 3. Resolve and price 'Dedicated' pool.
* 4. Resolve and price 'Common' pool.
*********************************************
*/
EXECUTE usp_PSPriceWASPCalcResolveDriver
@GasMonthx,@WhichVolumex,@yCIDEntity,'None'
EXECUTE usp_PSPriceAll
@GasMonthx,0,@WhichVolumex,0,@yCIDEntity,'None'
EXECUTE usp_PSPriceCostAll
@GasMonthx,@WhichVolumex,@yCIDEntity,0,'None'
EXECUTE usp_PSPriceWASPDivieOutProceedsN
@GasMonthx,@WhichVolumex,@yCIDEntity
EXECUTE usp_PSPriceWASPCalcResolveDriver
@GasMonthx,@WhichVolumex,@yCIDEntity,'Dedicated'
EXECUTE usp_PSPriceAll
@GasMonthx,0,@WhichVolumex,0,@yCIDEntity,'Dedicated'
EXECUTE usp_PSPriceCostAll
@GasMonthx,@WhichVolumex,@yCIDEntity,0,'Dedicated'
EXECUTE usp_PSPriceWASPCalcResolveDriver
@GasMonthx,@WhichVolumex,@yCIDEntity,'Common'
EXECUTE usp_PSPriceAll
@GasMonthx,0,@WhichVolumex,0,@yCIDEntity,'Common'
EXECUTE usp_PSPriceCostAll
@GasMonthx,@WhichVolumex,@yCIDEntity,0,'Common'
/*
*********************************************
* Calculate Transport contract gas inventory
* items (create them along with any
* transport deals).
*********************************************
EXECUTE usp_PSPriceTransportAll
@GasMonthx,@WhichVolumex,0,@yCIDEntity
*/
/*
*********************************************
* Indicate that the gas month is finished
* and commit the updates.
*********************************************
*/
UPDATE
        rGasMonth
        SET
                LockedUser=''
```

```
                                                WHERE
                                                        GasMonth=@GasMonthx
                                        /*
                                        ********************************************
                                        * Check to make sure that any items that
                                        * require an invoice number gets created.
                                        * This is only applicable when the gas month
                                        * is in an 'Invoiced' state already. This
                                        * picks up any new deals/meters created
                                        * after the gas month promoted to 'Invoiced'.
                                        ********************************************
                                        */
                                        IF (@yCurrentStatus='Invoiced')
                                                BEGIN
                                                EXECUTE usp_PSPriceAnyNewInvoicesNeeded
@yGasMonth,@yCIDEntity
                                                END
                                        FETCH NEXT FROM GasMonthCursor INTO
@yGasMonth,@yCurrentStatus
                        END
                CLOSE GasMonthCursor
                DEALLOCATE GasMonthCursor
                FETCH NEXT FROM EntityCIDCursor INTO @yCIDEntity
        END
CLOSE EntityCIDCursor
DEALLOCATE EntityCIDCursor
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceComponentsCheck(
                                                @PIDx INTEGER,
                                                @WhichPricex INTEGER,
                                                @GasMonthx DATETIME,
                                                @DBCRx INTEGER
                                                )
AS
BEGIN
/*
************************************************************************************
Name: usp_PSPriceComponentsCheck

Description:

Create any system generated pricing components automatically. Any existing
system generated pricing components are deleted.  Then they are recreated
within this particular process.  This procedure should be invoked for all
packages that were created within a given gas month. Current System
Generated Items include price components tagged as 'NETBACK PERCENTAGE' or
'WASP'.

Inputs:

PIDx - Package Identifier
WhichPricex - 0=Nominations, 1=Actuals
GasMonthx - Gas Month for Price Calculations
DBCRx - 0=Purchase, 1=Sales

History:
```

```
05/12/1999 JAMIE Original Creation.

07/28/2000 JAMIE Modify this process so that OIL, GAS or LIQUIDS is used when
obtaining the netback percentage.  This is based on the product ID for the deal.

08/17/2000 JAMIE Modify the process to eliminate any pricing entries on
WASP/EQUITY deals ('Common' pool).  This will ensure that the only pricing
entries on the wasp deals are those that are system generated.

*******************************************************************************************
*/
/*
*********************************************
* Declare all variables and cursors
* that are needed by this process.
*********************************************
*/
DECLARE @zProductID INTEGER
DECLARE @zProductNetbackType VARCHAR(12)
DECLARE @yWaspIndicator VARCHAR(10)
DECLARE @yEngineMasterRecords INTEGER
DECLARE @yEngineMasterETID_Key INTEGER
DECLARE @yEngineMasterPriceSequence INTEGER
DECLARE @yNetBackPercentage DECIMAL(19,8)

DECLARE ETIDCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    DISTINCT
                    ETID
          FROM
                    Engine_Master
          WHERE
                    PID=@PIDx
/*
*********************************************
*  Get the WASP indicator for this
*  particular deal via a function call.
*  This is based on how the deal is
*  classified.
*********************************************
*/
EXECUTE usp_fGetWaspIndicator @PIDx,@yWaspIndicator OUTPUT
/*
*********************************************
* All deals should have system generated
* price entries removed here...
*
* In addition, 'Common' wasp pool deals
* will have all non system generated
* price entries removed. Only purchase
* deals are impacted by system generated
* entries.
*********************************************
*/
OPEN ETIDCursor
FETCH NEXT FROM ETIDCursor INTO @yEngineMasterETID_Key
WHILE @@FETCH_STATUS = 0
          BEGIN
                    IF @yWaspIndicator='Common'
                              BEGIN
                                        IF @DBCRx=0
                                                  BEGIN
                                                            DELETE
                                                                      FROM
                                                                                Engine_MasterPrice
                                                                      WHERE

          (Engine_MasterPrice.ETID=@yEngineMasterETID_Key) AND
```

```
                (Engine_MasterPrice.NomOrActual=@WhichPricex)
                                        END
                        END
                ELSE
                        BEGIN
                                DELETE
                                        FROM
                                                Engine_MasterPrice
                                        WHERE

                (Engine_MasterPrice.ETID=@yEngineMasterETID_Key) AND

                (Engine_MasterPrice.NomOrActual=@WhichPricex) AND
                                                (Engine_MasterPrice.PriceTag=ANY(SELECT
PriceTag FROM PriceComponents WHERE SystemGenerated='Y'))
                        END
                FETCH NEXT FROM ETIDCursor INTO @yEngineMasterETID_Key
        END
CLOSE ETIDCursor
DEALLOCATE ETIDCursor
/*
*******************************************
* Now generate (insert) the price
* components that are required.  These
* system generated price components are
* recognized by the PriceTag name.  There
* should be a section within this procedure
* specifically for any system generated
* pricing components.
*******************************************
*/
/*
*******************************************
* "WASP" and "NETBACK PERCENTAGE"
*
* These two compononents go hand-in-hand.
*
* 1. Only system generate these if it is
* a purchase task and the deal is considered
* 'Wasp'able.
*******************************************
*/
IF ((@yWaspIndicator='Common') OR (@yWaspIndicator='Dedicated')) AND (@DBCRx=0)
        BEGIN
                /*
                *******************************************
                * Determine the correct product type in order
                * to get the correct contract netback
                * tier information.
                *******************************************
                */
                EXECUTE usp_fGetWaspType @PIDx,@zProductNetbackType OUTPUT
                /*
                *******************************************
                * Now go and find an Engine_Master record
                * to attach these components too.  If one
                * is not found, then insert one.  An
                * attempt to preserve the existing record
                * will ensure that nominations and pipe
                * line actuals will utilize the same
                * Engine_Master entity.
                *******************************************
                */
                SELECT @yEngineMasterRecords = ISNULL((SELECT count(*) FROM engine_master
                                                        WHERE PID=@PIDx
AND Effective=@GasMonthx AND STID=8 AND VolLevel=0),0)
                IF @yEngineMasterRecords=0
                        BEGIN
                                INSERT
```

```
                                        INTO
                                                Engine_Master
                (PID,Effective,STID,VolLevel,VolGroup,VarFixed,MMBtuMCF,TierThreshold)
                                        VALUES
                                                (@PIDx,@GasMonthx,8,0,@PIDx,1,1,1)
                        END
                SELECT @yEngineMasterETID_Key = ISNULL((SELECT MIN(ETID) FROM Engine_Master
                                                                WHERE PID=@PIDx
AND Effective=@GasMonthx AND STID=8 AND VolLevel=0),0)
                /*
                ******************************************
                * At this point we now either have a valid
                * ETID (key) to the Engine_Master or 0.
                * There should be only a single record on
                * the Engine_Master for these types of
                * packages.
                *
                * Now insert the 'WASP' price component.
                ******************************************
                */
                IF @yEngineMasterETID_Key > 0
                        BEGIN
                                SELECT @yEngineMasterPriceSequence = ISNULL((SELECT
MAX(SequenceNo) FROM Engine_MasterPrice

                WHERE ETID=@yEngineMasterETID_Key AND NomOrActual=@WhichPricex),0)
                                SELECT @yEngineMasterPriceSequence =
@yEngineMasterPriceSequence+1
                                INSERT
                                        INTO
                                                Engine_MasterPrice

        (ETID,PriceTag,OperandVariable,PriceVariable,CreateUser,CreateDate,LastUpdateUser,
                                                LastUpdateDate,SequenceNo,NomOrActual)
                                        VALUES

        (@yEngineMasterETID_Key,'WASP','+','WASP',UPPER(user_name()),

        getdate(),UPPER(user_name()),getdate(),@yEngineMasterPriceSequence,@WhichPricex)
                        END
                /*
                ******************************************
                * Now invoke the 'NETBACK PERCENTAGE'
                * calculation routine and then insert this
                * particular price component.  Remember to
                * put the netback percentage into its
                * 'string' representation.
                ******************************************
                */
                IF @yEngineMasterETID_Key > 0
                        BEGIN
                                EXECUTE usp_fGetNetbackPercentage
@PIDx,@GasMonthx,@zProductNetbackType,@WhichPricex,@yNetBackPercentage OUTPUT
                                IF @yNetBackPercentage IS NULL
                                        BEGIN
                                                SELECT @yNetBackPercentage = 0
                                        END
                                SELECT @yEngineMasterPriceSequence =
@yEngineMasterPriceSequence+1
                                INSERT
                                        INTO
                                                Engine_MasterPrice

        (ETID,PriceTag,OperandVariable,PriceVariable,CreateUser,

        CreateDate,LastUpdateUser,LastUpdateDate,SequenceNo,NomOrActual)
                                        VALUES
                                                (@yEngineMasterETID_Key,'NETBACK
PERCENTAGE','*',LTRIM(STR(@yNetBackPercentage,8,4)),
            UPPER(CURRENT_USER),getdate(),UPPER(CURRENT_USER),getdate(),@yEngineMasterPriceSequence --
,@WhichPricex)
                                    END
            END
END




GO
SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_PSPriceCost(
                                        @GasMonthx DATETIME,
                                        @WhichPricex INTEGER,
                                        @PKGx INTEGER,
                                        @STIDx INTEGER,
                                        @PCIDx INTEGER,
                                        @TIDx INTEGER,
                                        @CostLevelx VARCHAR(12),
                                        @CostBasisx VARCHAR(40),
                                        @CostRateOrAmountx DECIMAL(19,6),
                                        @TotalVolumex DECIMAL(19,2),
                                        @MeterVolumex DECIMAL(19,2)
                                        )
AS
BEGIN
/*
*******************************************************************************************
Name: usp_PSPriceCost

Description: This particular procedure will perform the actual calculations and post
updates to the engine table (for other costs associated with deals). This is done
for each meter within a deal for an other cost item.

Inputs:

GasMonthx (Gas Month to cost)
WhichPricex (0=Nominations, 1=Actualizations)
PKGx (deal id)
STIDx (engine transaction id)
PCIDx (deal other cost unique id (see PackageCosts table)
TIDx (gas inventory identifier)
CostLevelx (Level that cost is appropriated towards)
CostBasisx (rules governing calculation of the cost)
CostRateOrAmountx (rate or amount involved in cost)
TotalVolumex (total volume for deal)
MeterVolumex (total volume for meter within deal).

History:

10/20/99 JAMIE Initial creation.

03/26/00 JAMIE Modified to allow for zero volume deals to have other (fixed) costs
assigned to them.

10/03/20 JAMIE Modified to correct problem associated with 'METER'
calculations using entire deal volume.

12/01/2000 JAMIE Modified to apply the netback percentage to the other
cost when it is calculated. This percentage is only applicable to purchase
deals that are in the 'Common' or 'Dedicated' pools.
```

```
12/10/2000 JAMIE Modified to check for the apply netback flag on the
cost record in order to determine if the netback percentage should be
applied to the cost.

*****************************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zNetbackPercent DECIMAL(19,6)
DECLARE @zProductNetbackType VARCHAR(12)
DECLARE @yWaspIndicator VARCHAR(10)
DECLARE @zDBCR INTEGER
DECLARE @zApplyNetback VARCHAR(1)

DECLARE @zPercentToApply DECIMAL(19,4)
DECLARE @zAmountToApply DECIMAL(19,2)
DECLARE @zTotalSaleOrPurchValue DECIMAL(19,2)
DECLARE @zTotalMeters INTEGER

/*
*******************************************
* Initialize any fields required.
*******************************************
*/
SELECT @zNetbackPercent=0
SELECT @zAmountToApply=0
SELECT @zPercentToApply=1
SELECT @zTotalSaleOrPurchValue=0
/*
*******************************************
*  Get the WASP indicator for this
*  particular deal via a function call.
*  This is based on how the deal is
*  classified.
*******************************************
*/
EXECUTE usp_fGetWaspIndicator @PKGx,@yWaspIndicator OUTPUT
SELECT @zDBCR=ISNULL((SELECT packagedbcr FROM package WHERE pkg=@PKGx),0)
SELECT @zApplyNetback=ISNULL((SELECT applynetback from packagecosts WHERE pcid=@PCIDx),'Y')
/*
*******************************************
* Determine the correct product type in order
* to get the correct contract netback
* tier information.
*******************************************
*/
IF @zDBCR=0
          BEGIN
                    IF (@yWaspIndicator='Common') OR (@yWaspIndicator='Dedicated')
                              BEGIN
                                        EXECUTE usp_fGetWaspType @PKGx,@zProductNetbackType
OUTPUT
                                        EXECUTE usp_fGetNetbackPercentage
@PKGx,@GasMonthx,@zProductNetbackType,@WhichPricex,@zNetbackPercent OUTPUT
                              END
          END
/*
*******************************************
* Determine the percentage of whatever the
* cost will calculate to here.
* involved with this calculation. If it
* is a deal level fixed cost then show
* zeros IF there is no volume.
*******************************************
*/
```

```
IF (@MeterVolumex<>0) AND (@TotalVolumex<>0)
          BEGIN
                    IF @CostLevelx='DEAL'
                              BEGIN
                                        SELECT
@zPercentToApply=CONVERT(DECIMAL(19,4),@MeterVolumex)/CONVERT(DECIMAL(19,4),@TotalVolumex)
                              END
          END
IF (@MeterVolumex = 0) AND (@CostLevelx='DEAL')
          BEGIN
                    SELECT @zPercentToApply=0
          END
/*
*******************************************
* If the cost is a FIXED AMOUNT and there
* is no volume for the deal then determine
* the amount to apply based on the number
* of meters involved in the deal.  If 1
* meter only then 100% of cost assessed to
* that meter.  If 2 meters then 50% assessed
* to each one. etc..
*******************************************
*/
IF (@MeterVolumex=0) AND (@TotalVolumex=0)
          BEGIN
                    IF @CostBasisx='Fixed Amount'
                              BEGIN
                                        SELECT @zTotalMeters=ISNULL((SELECT count(*) FROM GasInv
WHERE PKG=@PKGx AND GasMonth=@GasMonthx),0)
                                        IF @zTotalMeters <> 0
                                                  BEGIN
                                                            SELECT
@zPercentToApply=(1/CONVERT(DECIMAL(19,4),@zTotalMeters))
                                                            SELECT
@zAmountToApply=(@CostRateOrAmountx*@zPercentToApply)
                                                  END
                              END
          END
/*
*******************************************
* Calculate based on fixed amount
* here...  Since this is a fixed amount
* then the amount should be calculated
* proportionately based on the total
* volume percentage to the deal.
*******************************************
*/
IF @CostBasisx='Fixed Amount'
          BEGIN
                    IF (@CostRateOrAmountx<>0) AND (@zPercentToApply<>0)
                              BEGIN
                                        SELECT
@zAmountToApply=(@CostRateOrAmountx*@zPercentToApply)
                              END
          END
/*
*******************************************
* Calculate based on a rate applied
* against MMBTU's here... Regardless
* of whether or not this is a 'DEAL'
* level or 'METER' level charge the
* cost should be based on meter
* volume.
*******************************************
*/
IF (@MeterVolumex<>0)
          BEGIN
                    IF @CostBasisx='Rate Applied to MMBTUs'
                              BEGIN
                                        IF (@CostRateOrAmountx<>0)
                                        BEGIN
                                                  SELECT
@zAmountToApply=((CONVERT(DECIMAL(19,4),@MeterVolumex)*@CostRateOrAmountx))
                                        END
                              END
          END
/*
*******************************************
* Calculate based on the total dollar amount
* previously calculated here... Since
* this particular cost is calculating on
* just the amount for the associated
* meter (ie.. sum of engine based on
* TID) then the 'PercentToApply' is
* not applicable.
*******************************************
*/
IF (@MeterVolumex<>0) AND (@TotalVolumex<>0)
          BEGIN
                    IF @CostBasisx='Rate Applied to Value'
                              BEGIN
                                        IF @WhichPricex=0
                                        BEGIN
                                                  SELECT
@zTotalSaleOrPurchValue=ISNULL((SELECT SUM(amount) FROM engine WHERE tid=@tidx AND (stid=8 OR
stid=9)),0)
                                        END
                                        IF @WhichPricex=1
                                        BEGIN
                                                  SELECT
@zTotalSaleOrPurchValue=ISNULL((SELECT SUM(amountact) FROM engine WHERE tid=@tidx AND (stid=8 OR
stid=9)),0)
                                        END
                                        if(@CostRateOrAmountx<>0) AND (@zTotalSaleOrPurchValue<>0)
                                        BEGIN
                                                  SELECT
@zAmountToApply=(@zTotalSaleOrPurchValue*@CostRateOrAmountx)
                                        END
                              END
          END
/*
*******************************************
* Finally, post the cost amount to the
* Engine table. If the engine table for
* this transaction does not yet exist then
* insert it, otherwise just update it...
*
* Make sure that actual calculations and
* nomination calculations are done within
* their respective 'buckets'.
*******************************************
*/
/*
*************************************
* First apply the netback if it
* is there AND if the apply
* netback flag has been set
* on the cost item.
*************************************
*/
IF @zApplyNetback = 'Y'
          BEGIN
                    IF @zNetbackPercent<>0
                              BEGIN
                                        SELECT
@zAmountToApply=ROUND((@zAmountToApply*@zNetbackPercent),2)
                              END
          END
/*
*************************************
```

```
* Apply and post the amount
* here...
***************************************
*/
IF @WhichPricex=0
          BEGIN
                    IF (SELECT count(*) FROM Engine WHERE TID=@TIDx AND STID=@STIDx AND
Effective=@GasMonthx AND VolLevel=0)=0
                              BEGIN
                                        INSERT
                                                  INTO
                                                            Engine

          (TID,STID,Effective,VolLevel,VolGroup,MMBTuMCF,Volume,Amount,PriceOrRateNom,PriceOrRateAct,Volu
meAct,AmountAct,EM_ETID)
                                                  VALUES

          (@TIDx,@STIDx,@GasMonthx,0,@PKGx,1,0,ROUND(@zAmountToApply,2),0,0,0,0,@PCIDx)
                              END
                    ELSE
                              BEGIN
                                        UPDATE
                                                  engine
                                                  SET

          Amount=Amount+ROUND(@zAmountToApply,2)
                                                  WHERE
                                                            TID=@TIDx AND
                                                            STID=@STIDx AND
                                                            Effective=@GasMonthx AND
                                                            VolLevel=0
                              END
          END
IF @WhichPricex=1
          BEGIN
                    IF (SELECT count(*) FROM Engine WHERE TID=@TIDx AND STID=@STIDx AND
Effective=@GasMonthx AND VolLevel=0)=0
                              BEGIN
                                        INSERT
                                                  INTO
                                                            Engine

          (TID,STID,Effective,VolLevel,VolGroup,MMBTuMCF,Volume,Amount,PriceOrRateNom,PriceOrRateAct,Volu
meAct,AmountAct,EM_ETID)
                                                  VALUES

          (@TIDx,@STIDx,@GasMonthx,0,@PKGx,1,0,0,0,0,0,ROUND(@zAmountToApply,2),@PCIDx)
                              END
                    ELSE
                              BEGIN
                                        UPDATE
                                                  engine
                                                  SET

          AmountAct=AmountAct+ROUND(@zAmountToApply,2)
                                                  WHERE
                                                            TID=@TIDx AND
                                                            STID=@STIDx AND
                                                            Effective=@GasMonthx AND
                                                            VolLevel=0
                              END
          END
END


GO
```

```
SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER OFF   SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_PSPriceCostAll(
                                        @GasMonthx DATETIME,
                                        @WhichPricex INTEGER,
                                        @EntityCIDx VARCHAR(12),
                                        @DBCRx INTEGER,
                                        @IncludeInWaspx VARCHAR(10)
                                        )
AS
BEGIN
/*
***************************************************************************************
Name: usp_PSPriceCostAll

Description: Loop thruough all other costs associated to deals within a given month
then apply the cost to the dean (posting engine records reflecting the cost amounts).
or sale) and invoke the price procedures.

Inputs:

GasMonthx - Gas Month to price),
WhichPricex - 0=Nominations, 1=Actualizations
EntityCIDx - owning entry company identifier
DBCRx - 0=Purchases, 1=Sales (deals)
IncludeInWaspx = '' for all or specific pool (ie. 'Common', etc.).

History:

10/20/99 JAMIE Initial creation.

03/26/00 JAMIE Modified to allow for zero volume deals to have other (fixed) costs
assigned to them.

05/24/2000 JAMIE Modified to make sure that the calculation was within a specific
entity.

10/03/2000 JAMIE Modified to accept two additional parameters to dictate which
pool and whether or not purchases or sales were to be calculated upon...

***************************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @zTotalVolume DECIMAL(19,2)
DECLARE @zMeterVolume DECIMAL(19,2)
DECLARE @zVolumeStatus INTEGER
DECLARE @zPriceStatus INTEGER
DECLARE @zIncludeInWasp VARCHAR(10)

DECLARE @yPCID INTEGER
DECLARE @yPKG INTEGER
DECLARE @ySTID INTEGER
DECLARE @yCostLevel VARCHAR(12)
DECLARE @yCostMID INTEGER
DECLARE @yCostBasis VARCHAR(40)
DECLARE @yCostRateOrAmount DECIMAL(19,4)

DECLARE @wTID INTEGER
DECLARE @wNom DECIMAL(19,2)
DECLARE @wPipelineActuals DECIMAL(19,2)
DECLARE @wGasInv_MID INTEGER

DECLARE @eETID INTEGER
DECLARE @eVolume DECIMAL(19,2)
DECLARE @ePriceOrRateNom DECIMAL(19,6)
DECLARE @eVolumeAct DECIMAL(19,2)
DECLARE @ePriceOrRateAct DECIMAL(19,6)
DECLARE @evolumestatus INTEGER
DECLARE @epricestatus INTEGER
DECLARE @ePKG INTEGER

DECLARE PackageCostsCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                PackageCosts.PCID,
                PackageCosts.PKG,
                PackageCosts.STID,
                PackageCosts.CostLevel,
                PackageCosts.CostMID,
                PackageCosts.CostBasis,
                PackageCosts.CostRateOrAmount
                FROM
                        PackageCosts
                WHERE
                        PackageCosts.PKG=ANY(SELECT PKG FROM Package,k WHERE
PackageGasMonth=@GasMonthx AND
                                                                        K.KID=Package.KID AND
K.EntityCID=@EntityCIDx AND Package.PackageDBCR=@DBCRx)
                ORDER BY
                        PackageCosts.PKG,
                        PackageCosts.STID

DECLARE EngineCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                engine.etid,
                engine.volume,
                engine.priceorratenom,
                engine.volumeact,
                engine.priceorrateact,
                engine.volumestatus,
                engine.pricestatus,
                package.pkg
        FROM
                engine,
                gasinv,
                package,
                k
        WHERE
                package.pkg=gasinv.pkg AND
                k.kid=package.kid AND
                k.entitycid=@entitycidx AND
                gasinv.gasmonth=@GasMonthx AND
                engine.tid=gasinv.tid AND
                gasinv.pricetype=1 AND
                gasinv.dbcr=@DBCRx
/*
*******************************************
* Loop through each other package cost
* involved with this calculation.
*******************************************
*/
SELECT @zMessage = 'PSPriceCostAll Running To Calculate Other Costs for all Deals'
EXECUTE usp_Message @zMessage
OPEN PackageCostsCursor
FETCH NEXT FROM PackageCostsCursor INTO
@yPCID,@yPKG,@ySTID,@yCostLevel,@yCostMID,@yCostBasis,@yCostRateOrAmount
WHILE @@FETCH_STATUS = 0
        BEGIN
                BEGIN TRANSACTION
                /*
                *******************************************
```

```
                    * Sum the appropriate volumes for this
                    * deal depending on whether nominations are
                    * being calculated OR pipeline actuals are
                    * begin calculated.
                    ********************************************
                    */
                    SELECT @zMessage =  'PSPriceCostAll Calculating Costs for Deal...' + CAST(@yPKG AS
VARCHAR(10))
                    EXECUTE usp_Message @zMessage
                    EXECUTE usp_fGetWaspIndicator @yPKG,@zIncludeInWasp OUTPUT
                    IF (@IncludeInWaspx='') OR (@IncludeInWaspx=@zIncludeInWasp)
                        BEGIN
                            IF @WhichPricex=0
                                BEGIN
                                    SELECT @zTotalVolume=ISNULL((SELECT
SUM(Nom) FROM GasInv WHERE GasInv.PKG=@yPKG AND GasInv.PriceType=1),0)
                                END
                            IF @WhichPricex=1
                                BEGIN
                                    SELECT @zTotalVolume=ISNULL((SELECT
SUM(PipelineActuals) FROM GasInv WHERE GasInv.PKG=@yPKG AND GasInv.PriceType=1),0)
                                END
                            /*
                            ********************************************
                            * Open a cursor on all meters associated
                            * with this deal.
                            ********************************************
                            */
                            DECLARE GasInvCursor CURSOR LOCAL STATIC
FORWARD_ONLY FOR
                                SELECT
                                    GasInv.TID,
                                    GasInv.Nom,
                                    GasInv.PipelineActuals,
                                    GasInv.GasInv_MID
                                    FROM
                                        GasInv
                                    WHERE
                                        GasInv.PKG=@yPKG AND
                                        GasInv.PriceType=1
                            OPEN GasInvCursor
                        FETCH NEXT FROM GasInvCursor INTO
@wTID,@wNom,@wPipelineActuals,@wGasInv_MID
                            WHILE @@FETCH_STATUS = 0
                                BEGIN
                                /*
                                ********************************************
                                * Depending on which pricing routine is
                                * run, set the appropriate meter volume
                                * field.
                                ********************************************
                                */
                                IF @WhichPricex=0
                                        BEGIN
                                        SELECT
@zMeterVolume=@wNom
                                        END
                                IF @WhichPricex=1
                                        BEGIN
                                        SELECT
@zMeterVolume=@wPipelineActuals
                                        END
                                /*
                                ********************************************
                                * Invoke the detail cost routine in order
                                * to calculate and post the cost totals
                                * to the Engine Database.
                                ********************************************
                                */
```

```
                                                  IF (@yCostLevel='DEAL') OR (@yCostLevel='METER'
AND @yCostMID=@wGasInv_MID)
                                                              BEGIN
                                                              EXECUTE usp_PSPriceCost
@GasMonthx,@WhichPricex,@yPKG,@ySTID,@yPCID,

        @wTID,@yCostLevel,@yCostBasis,@yCostRateOrAmount,

        @zTotalVolume,@zMeterVolume
                                                              END
                                                  FETCH NEXT FROM GasInvCursor INTO
@wTID,@wNom,@wPipelineActuals,@wGasInv_MID
                                                  END
                                          CLOSE GasInvCursor
                                          DEALLOCATE GasInvCursor
                                  END
                          COMMIT WORK
                          FETCH NEXT FROM PackageCostsCursor INTO
@yPCID,@yPKG,@ySTID,@yCostLevel,@yCostMID,@yCostBasis,@yCostRateOrAmount
        END
CLOSE PackageCostsCursor
DEALLOCATE PackageCostsCursor
/*
*******************************************
* Loop through and set the status flags
* on the engine record IF the price or
* volumes or amounts are different
* between noms and actuals.  Make
* sure the logic exists to only calculate
* those deals (purchases or sales)
* within the correct WASP pool.
*******************************************
*/
IF @WhichPricex=1
        BEGIN
                SELECT @zMessage = 'PSPriceCostAll Running To Set Price & Volume Variance Status
Indicators...'
                EXECUTE usp_Message @zMessage
                OPEN EngineCursor
                FETCH NEXT FROM EngineCursor INTO
@eETID,@eVolume,@ePriceOrRateNom,@eVolumeAct,@ePriceOrRateAct,@eVolumeStatus,@ePriceStatus,@ePKG
                WHILE @@FETCH_STATUS = 0
                        BEGIN
                                EXECUTE usp_fGetWaspIndicator @ePKG,@zIncludeInWasp
OUTPUT
                                IF (@IncludeInWaspx='') OR (@IncludeInWaspx=@zIncludeInWasp)
                                        BEGIN
                                        /*
                                        *******************************************
                                        * Check prices and volumes here.
                                        *******************************************
                                        */
                                        SELECT @zVolumeStatus=0
                                        SELECT @zPriceStatus=0
                                        IF @eVolume<>@eVolumeAct
                                                BEGIN
                                                SELECT @zVolumeStatus=1
                                                END
                                        IF @ePriceOrRateNom<>@ePriceOrRateAct
                                                BEGIN
                                                SELECT @zPriceStatus=1
                                                END
                                        IF (@zVolumeStatus<>@eVolumeStatus) OR
(@zPriceStatus<>@ePriceStatus)
                                                BEGIN
                                                UPDATE
                                                        engine
                                                        SET

        volumestatus=@zVolumeStatus,
                pricestatus=@zPriceStatus                                                                --
                                                                                                   WHERE  -

                ETID=@eETID
                                                                               END
                                                               END
                                                     FETCH NEXT FROM EngineCursor INTO
@eETID,@eVolume,@ePriceOrRateNom,@eVolumeAct,@ePriceOrRateAct,@eVolumeStatus,@ePriceStatus,@ePKG
                                            END
                          CLOSE EngineCursor
                          DEALLOCATE EngineCursor
                END
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceCreateActualEntries(
                                                            @GasMonthx DATETIME
                                                                        )
AS
BEGIN
SET NOCOUNT ON
/*
************************************************************************************
Name: usp_PSPriceCreateActualEntries

Description: This routine will clear out any existing links and pricing enties
that may have already been setup for pipeline actuals.  It will then copy the
nominiation pricing and linking entries for pipeline actuals (within the given
month).  This process should only get invoked with the status of a given month
within the gas control system goes from 'Sales' to 'Invoiced'  at that point in time
the accounting group will be responsible for any further modifications.

Inputs:

GasMonthx (Gas Month to calculate),

History:

08/04/1999 JAMIE   Original creation

08/25/2000 JAMIE Modified to remove the PackageLinks delete and build
logic (replaced by new routing structures).

************************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @yPKG INTEGER
DECLARE @yETID INTEGER
DECLARE @yEM_ETID INTEGER
/*
*******************************************
* Clear out the link and price entry
* structures for the specified month
* here... These entries will be
* recreated (from Nom side) in the
* next step.
*
* Database triggers take care of the
* individual pricing components in
* the Engine_MasterPrice table.
*********************************************
*/
SELECT @zMessage = 'PSPriceCreateActualEntries, removing Engine_MasterPrice...'
EXECUTE usp_Message @zMessage
DECLARE Engine_MasterDeleteCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    DISTINCT
                    (Engine_Master.ETID)
                    FROM
                              Engine_Master,
                              GasInv,
                              Engine_MasterPrice
                    WHERE
                              GasInv.GasMonth=@GasMonthx AND
                              GasInv.PriceType=1 AND
                              GasInv.PKG=Engine_Master.PID AND
                              Engine_MasterPrice.ETID=Engine_Master.ETID AND
                              Engine_MasterPrice.NomOrActual=1
OPEN Engine_MasterDeleteCursor
FETCH NEXT FROM Engine_MasterDeleteCursor INTO @yEM_ETID
WHILE @@FETCH_STATUS = 0
          BEGIN
                    BEGIN TRANSACTION
                    SELECT @zMessage = 'PSPriceCreateActualEntries, actual Engine_MasterPrice removed...'
                    EXECUTE usp_Message @zMessage
                    DELETE
                              FROM
                                        Engine_MasterPrice
                              WHERE
                                        ETID=@yEM_ETID AND
                                        NomOrActual=1
                    COMMIT WORK
                    FETCH NEXT FROM Engine_MasterDeleteCursor INTO @yEM_ETID
          END
CLOSE Engine_MasterDeleteCursor
DEALLOCATE Engine_MasterDeleteCursor
/*
*********************************************
* Now bulk populate the engine
* pricing information. Taking nom
* pricing entries and creating actual
* pricing entries.
*********************************************
*/
SELECT @zMessage = 'PSPriceCreateActualEntries, running GasInv cursor...'
EXECUTE usp_Message @zMessage
DECLARE GasInvCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    DISTINCT
                    (GasInv.PKG)
                    FROM
                              GasInv
                    WHERE
                              GasInv.GasMonth=@GasMonthx AND
                              GasInv.PriceType=1
OPEN GasInvCursor
FETCH NEXT FROM GasInvCursor INTO @yPKG
WHILE @@FETCH_STATUS = 0
          BEGIN
                    BEGIN TRANSACTION
                    SELECT @zMessage = 'PSPriceCreateActualEntries, obtaining price entries for GasInv
Package...'
```

```
                    EXECUTE usp_Message @zMessage
                    DECLARE Engine_MasterCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                            SELECT
                                    DISTINCT
                                    (ETID)
                                    FROM
                                            Engine_Master
                                    WHERE
                                            PID=@yPKG
                    OPEN Engine_MasterCursor
                    FETCH NEXT FROM Engine_MasterCursor INTO @yETID
                    WHILE @@FETCH_STATUS = 0
                            BEGIN
                                    SELECT @zMessage =  'PSPriceCreateActualEntries, inserting actual
prices .'
                                    EXECUTE usp_Message @zMessage
                                    INSERT
                                            INTO
                                                    Engine_MasterPrice

        (ETID,PriceTag,OperandVariable,PriceVariable,CreateUser,

        CreateDate,LastUpdateUser,LastUpdateDate,SequenceNo,NomOrActual)
                                            (SELECT
ETID,PriceTag,OperandVariable,PriceVariable,CreateUser,CreateDate,LastUpdateUser,LastUpdateDate,
                                                    SequenceNo,1 FROM Engine_MasterPrice
WHERE ETID=@yETID AND NomOrActual=0)
                                    FETCH NEXT FROM Engine_MasterCursor INTO @yETID
                            END
                    CLOSE Engine_MasterCursor
                    DEALLOCATE Engine_MasterCursor
                    COMMIT WORK
                    FETCH NEXT FROM GasInvCursor INTO @yPKG
        END
CLOSE GasInvCursor
DEALLOCATE GasInvCursor
END



GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceMarkActualAdjustments(
                                                    @GasMonthx DATETIME
                                                    )

AS
BEGIN
SET NOCOUNT ON
/*
**************************************************************************************************
Name: usp_PSPriceMarkActualAdjustments

Description: This routine will go through each inventory (and engine
records) in order to identify and mark those records that had some sort of
an actualization adjustment (price or volume).  The invoice number for sales
will get reset to a 'A' (last character) if it currently exists as an 'N'.

Inputs:

GasMonthx (Gas Month to calculate),

History:
```

```
12/15/1999 JAMIE   Original creation
******************************************************************************************
*/
/*
********************************************
* Declare all variables and cursors
* that are needed by this process.
********************************************
*/
DECLARE @zMessage VARCHAR(254)

DECLARE @yAcctgIdentifier VARCHAR(12)
DECLARE @zAcctgIdentifier VARCHAR(12)
DECLARE @zLastChar VARCHAR(1)
DECLARE @zInvoiceLength INTEGER

DECLARE @qTID INTEGER

/*
********************************************
* First set the modified by actuals flag
* across the board for all gasinventory
* items that have a price type of '1'
* (this includes 'Other Costs'.
*
* The defaults is set to 'N' then go
* and override with changes.
********************************************
*/
SELECT @zMessage = '**** STARTED PSPriceMarkActualAdjustments'
EXECUTE usp_Message @zMessage
DECLARE GasInv1Cursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                TID
                FROM
                        GasInv
                WHERE
                        GasMonth=@GasMonthx AND
                        PriceType = 1
OPEN GasInv1Cursor
FETCH NEXT FROM GasInv1Cursor INTO @qTID
WHILE @@FETCH_STATUS = 0
        BEGIN
                BEGIN TRANSACTION
                UPDATE
                        GasInv
                        SET
                                ModifiedByActuals='N'
                        WHERE
                                TID = @qTID
                COMMIT WORK
                FETCH NEXT FROM GasInv1Cursor INTO @qTID
        END
CLOSE GasInv1Cursor
DEALLOCATE GasInv1Cursor
/*
********************************************
* At this point all of the gas inventory
* items that have had some sort of
* modification done on them between
* noms and actuals will have been
* updated to a 'Y'.  Now go and reset
* the accounting identifier for each of
* these records.
********************************************
*/
SELECT @zMessage = 'PSPriceMarkActualAdjustments, make any modifiications'
EXECUTE usp_Message @zMessage
DECLARE GasInv2Cursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
```

```
          SELECT
                    DISTINCT
                    (G.AcctgIdentifier)
                    FROM
                              GasInv AS G,
                              Engine AS E
                    WHERE
                              GasMonth=@GasMonthx AND
                              G.PriceType=1 AND
                              E.TID=G.TID AND
                              (E.PriceStatus<>0 OR E.VolumeStatus<>0)
OPEN GasInv2Cursor
FETCH NEXT FROM GasInv2Cursor INTO @yAcctgIdentifier
WHILE @@FETCH_STATUS = 0
          BEGIN
                    BEGIN TRANSACTION
                    /*
                    ******************************************
                    * Make sure that it is a valid 6 digit
                    * invoice number AND the sixth digit
                    * contains an 'N' (for noms).
                    * Update all if this criteria has been
                    * met.
                    ******************************************
                    */
                    SELECT @zInvoiceLength=LEN(RTRIM(LTRIM(@yAcctgIdentifier)))
                    IF @zInvoiceLength=6
                              BEGIN
                                        SELECT @zAcctgIdentifier=RTRIM(LTRIM(@yAcctgIdentifier))
                                        SELECT @zLastChar=RIGHT(@zAcctgIdentifier,1)
                                        IF @zLastChar='N'
                                                  BEGIN
                                                  SELECT @zAcctgIdentifier=LEFT(@zAcctgIdentifier,5)+'A'
                                                  UPDATE
                                                                      GasInv
                                                                      SET
                                                                      ModifiedByActuals='Y',

          AcctgIdentifier=@zAcctgIdentifier
                                                            WHERE

          GasMonth=@GasMonthx AND

          AcctgIdentifier=@yAcctgIdentifier
                                                  END
                              END
                    COMMIT WORK
                    FETCH NEXT FROM GasInv2Cursor INTO @yAcctgIdentifier
          END
CLOSE GasInv2Cursor
DEALLOCATE GasInv2Cursor
SELECT @zMessage = '**** FINISHED PSPriceMarkActualAdjustments'
EXECUTE usp_Message @zMessage
END



GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPricePopulateEngine(
                                                            @PIDx INTEGER,
                                                            @WhichPricex INTEGER,
                                                            @GasMonthx DATETIME,
                                        @DBCRx INTEGER
                                        )
AS
BEGIN
/*
****************************************************************************************
Name: usp_PSPricePopulateEngine

Description:  Make sure that the price entries are populated on the engine
master for the respective Engine_Master pricing records.  Only those
engine records with pricetype=1 (STID=8 or STID=9) will be manipulated by
this process.

Inputs:
PIDx (unique Package ID)
WhichPricex(0=Nomination, 1=Actual)
GasMonthx (gas month calculation applicable toward)
DBCRx (0=Purchase,1=Sale)

History:

05/14/99  JAMIE  Original Creation

09/27/99  JAMIE  Modify this process to check for index baskets AND IF the
index basket is comprised of daily indices then populate the entire month
with the basket information.

01/27/00  JAMIE  Modified to delete engine records while in the actuals phase
if the nom information is zero (all nom information, price, volume, amount, etc.).

****************************************************************************************
*/
/*
**********************************************
* Declare all variables and cursors
* that are needed by this process.
**********************************************
*/
DECLARE @dPriceEntryType VARCHAR(12)
DECLARE @dPriceVariable VARCHAR(15)
DECLARE @tmpDailyIndexCount INTEGER
DECLARE @tmpUseEffective DATETIME
DECLARE @tmpEndDate DATETIME
DECLARE @tmpPrevEffective DATETIME
DECLARE @tmpNextEffectiveDate DATETIME
DECLARE @tmpNumberDays DATETIME
DECLARE @tmpVolumeInPeriod DECIMAL(19,2)
DECLARE @tmpDateToUse DATETIME
DECLARE @tmpEndEffectiveDate DATETIME
DECLARE @yETID INTEGER
DECLARE @yEffective DATETIME
DECLARE @ySTID INTEGER
DECLARE @yVolLevel INTEGER
DECLARE @yVolGroup INTEGER
DECLARE @yVarFixed INTEGER
DECLARE @yMMBtuMCF INTEGER
DECLARE @yTierThreshold INTEGER

DECLARE @yTID INTEGER

/*
**********************************************
* First, Go out and delete entries off the
* engine database related to this particular
* package.  If the pipeline actuals are
* being processed then just go and
* initialize any existing Engine record
* 'Actual' buckets to zero (leave the
* preexisting engine records intact).
*
```

```
* Modified on 01/27/2000 to delete engine
* records off actuals IF there are no nom
* numbers stored on the records...
*********************************************
*/
IF @WhichPricex=0
          BEGIN
                    DELETE
                              FROM
                                        Engine
                              WHERE
                                        TID=ANY(SELECT TID FROM GasInv WHERE PKG=@PIDx AND
PriceType=1 AND DBCR=@DBCRx)
          END
IF @WhichPricex=1
          BEGIN
                    DELETE
                              FROM
                                        Engine
                              WHERE
                                        TID=ANY(SELECT TID FROM GasInv WHERE PKG=@PIDx AND
PriceType=1 AND DBCR=@DBCRx) AND
                                        PriceOrRateNom=0 AND
                                        Volume=0 AND
                                        Amount=0
                    UPDATE
                              Engine
                              SET
                                        PriceOrRateAct=0,
                                        VolumeAct=0,
                                        AmountAct=0
                              WHERE
                                        TID=ANY(SELECT TID FROM GasInv WHERE PKG=@PIDx AND
PriceType=1 AND DBCR=@DBCRx)
          END
/*
*********************************************
* First, do a loop on all of the
* Engine_Master records in order to
* remove any that don't have any price
* records associated to it... (Orphans)..
* A commit point is placed here in order to
* insure that subsequent cursor activity
* only picks up valid price records.
*********************************************
*/
DECLARE Engine_MasterCursor1 CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    em.ETID,
                    em.Effective,
                    em.STID,
                    em.VolLevel,
                    em.VolGroup,
                    em.VarFixed,
                    em.MMBtuMCF,
                    em.TierThreshold
                    FROM
                              Engine_Master AS em
                    WHERE
                              (em.PID=@PIDx)
                    ORDER BY
                              em.Effective
OPEN Engine_MasterCursor1
FETCH NEXT FROM Engine_MasterCursor1 INTO
@yETID,@yEffective,@ySTID,@yVolLevel,@yVolGroup,@yVarFixed,@yMMBtuMCF,@yTierThreshold
WHILE @@FETCH_STATUS = 0
          BEGIN
                    IF ISNULL((SELECT count(*) FROM Engine_MasterPrice WHERE ETID=@yETID),0) < 1
                              BEGIN
                                        DELETE
                                        FROM
                                                  Engine_Master
                                        WHERE
                                                  ETID=@yETID
                              END
                    FETCH NEXT FROM Engine_MasterCursor1 INTO
@yETID,@yEffective,@ySTID,@yVolLevel,@yVolGroup,@yVarFixed,@yMMBtuMCF,@yTierThreshold
          END
CLOSE Engine_MasterCursor1
DEALLOCATE Engine_MasterCursor1
/*
*******************************************
* Now loop through the existing
* Engine_Master records.  These are the
* actual price entries that were input
* by the user. There can be a record
* PER DAY or a single record for the
* entire month.  Only 1 entry PER
* Effective date will be stored within
* the Engine table.  That is why the
* tmpPrevEffective is used within the
* cursor process.
*******************************************
*/
SELECT @tmpPrevEffective='01-01-1900'
DECLARE Engine_MasterCursor2 CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    em.ETID,
                    em.Effective,
                    em.STID,
                    em.VolLevel,
                    em.VolGroup,
                    em.VarFixed,
                    em.MMBtuMCF,
                    em.TierThreshold
                    FROM
                              Engine_Master AS em
                    WHERE
                              (em.PID=@PIDx)
                    ORDER BY
                              em.Effective
OPEN Engine_MasterCursor2
FETCH NEXT FROM Engine_MasterCursor2 INTO
@yETID,@yEffective,@ySTID,@yVolLevel,@yVolGroup,@yVarFixed,@yMMBtuMCF,@yTierThreshold
WHILE @@FETCH_STATUS = 0
          BEGIN
                    /*
                    *******************************************
                    * Check for daily index entries... If they
                    * are found then go and calculate the
                    * end date for which to insert engine
                    * records (automating a daily price
                    * entry to the engine for each day of
                    * the month up thru the end of the month
                    * or to the next effective date.
                    *
                    * This will also check for index basket
                    * monthly entries.  If the index basket
                    * contains daily indices then populate
                    * each day of the month just as if it
                    * was a daily index.
                    *******************************************
                    */
                    IF @yEffective<>@tmpPrevEffective
                              BEGIN
                                        EXECUTE usp_fLastDay @GasMonthx,@tmpEndDate OUTPUT
                                        SELECT @tmpDailyIndexCount=0
                                        DECLARE DailyCheckCursor CURSOR LOCAL STATIC
FORWARD_ONLY FOR
                                                  SELECT
                                        p.PriceEntryType,
                                        emp.PriceVariable
                                        FROM
                                                Engine_MasterPrice AS emp,
                                                PriceComponents AS p
                                        WHERE
                                                (emp.ETID=@yETID) AND

    (emp.NomOrActual=@WhichPricex) AND
                                                (p.PriceTag=emp.PriceTag) AND
                                                (p.PriceEntryType='Daily IDX' OR
p.PriceEntryType='Basket IDX')
                        OPEN DailyCheckCursor
                    FETCH NEXT FROM DailyCheckCursor INTO @dPriceEntryType,@dPriceVariable
                        WHILE @@FETCH_STATUS = 0
                                BEGIN
                                IF (@dPriceEntryType='Daily IDX') AND
(@tmpDailyIndexCount=0)
                                                BEGIN
                                                SELECT @tmpDailyIndexCount=1
                                                END
                                IF (@dPriceEntryType='Basket IDX') AND
(@tmpDailyIndexCount=0)
                                                BEGIN
                                                SELECT @tmpDailyIndexCount =
ISNULL((SELECT count(*) FROM IndexBasketLink,IndexRef

                    WHERE (IndexBasketLink.IndexBasketID=@dPriceVariable) AND

                                (IndexRef.IndexID=IndexBasketLink.IndexID) AND

                                (IndexRef.DailyIndex=1)),0)
                                                END
                                FETCH NEXT FROM DailyCheckCursor INTO
@dPriceEntryType,@dPriceVariable
                                END
                        CLOSE DailyCheckCursor
                        DEALLOCATE DailyCheckCursor
                        IF @tmpDailyIndexCount=0
                                BEGIN
                                        SELECT @tmpEndEffectiveDate=@yEffective
                                END
                        ELSE
                                BEGIN
                                        SELECT
@tmpEndEffectiveDate=ISNULL((SELECT DATEADD(day,-1,MIN(em.effective)) FROM Engine_Master AS em

    WHERE (em.PID=@PIDx) AND (em.Effective>@yEffective)),@tmpEndDate)
                                END
                        /*
                        *******************************************
                        * Now insert the new Engine records.
                        * These inserts will be based on a loop
                        * between the effective date from the
                        * Engine_Master record and the temp
                        * field tmpEndEffectiveDate.  This will
                        * provide for the 'proliferation' of
                        * daily index price entries (to the
                        * engine).  Only insert engine records
                        * if there is some sort of volume
                        * Nom or PipelineActual on associated
                        * with a specific day.
                        *
                        * If pipeline actuals then inserts do
                        * not automatically happen. A check
                        * is first made to see if the engine
                        * record is already there...
                        *******************************************
                        */
                        SELECT @tmpUseEffective=@yEffective
```

```
                                WHILE @tmpUseEffective <= @tmpEndEffectiveDate
                                        BEGIN
                                                DECLARE GasInventoryCursor CURSOR
LOCAL STATIC FORWARD_ONLY FOR
                                                        SELECT
                                                                DISTINCT
                                                                g.TID
                                                                FROM
                                                                        GasInv
AS g,
                                                                        GasInvD
AS gd
                                                                WHERE
        (gd.TID=g.TID) AND
        (g.PID=@PIDx) AND
        (g.GasMonth=@GasMonthx) AND
        (g.PriceType=1) AND
        (g.DBCR=@DBCRx) AND
        (gd.GasDay>=@tmpUseEffective) AND
        ((gd.Nom<>0) or(gd.PipelineActuals<>0))
                                        OPEN GasInventoryCursor
                                        FETCH NEXT FROM GasInventoryCursor INTO @yTID
                                                WHILE @@FETCH_STATUS = 0
                                                        BEGIN
                                                        IF (SELECT count(*) FROM
Engine WHERE TID=@yTID AND STID=@ySTID AND
                        Effective=@tmpUseEffective AND VolLevel=0)=0
                                                                        BEGIN
        INSERT
                INTO
                        Engine
                        (TID,STID,Effective,VolLevel,VolGroup,MMBtuMCF,EM_ETID)
                VALUES
                        (@yTID,@ySTID,@tmpUseEffective,0,@yVolGroup,@yMMBtuMCF,@yETID)
                                                                        END
                                                        ELSE
                                                                        BEGIN
        UPDATE
                Engine
                SET
                        EM_ETID=@yETID
                WHERE
                        TID=@yTID AND
                        STID=@ySTID AND
                        Effective=@tmpUseEffective AND
                        VolLevel=0
                                                                END
                                                                FETCH NEXT FROM
GasInventoryCursor INTO @yTID
                                                                END
                                                        CLOSE GasInventoryCursor
                                                        DEALLOCATE GasInventoryCursor
                                                SELECT
@tmpUseEffective=DATEADD(day,1,@tmpUseEffective)
                                                END
                                END
                        SELECT @tmpPrevEffective=@yEffective
                        FETCH NEXT FROM Engine_MasterCursor2 INTO
@yETID,@yEffective,@ySTID,@yVolLevel,@yVolGroup,@yVarFixed,@yMMBtuMCF,@yTierThreshold
                END
CLOSE Engine_MasterCursor2
DEALLOCATE Engine_MasterCursor2
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceTransportAll(
                                                @GasMonthx DATETIME,
                                                @WhichPricex INTEGER,
                                                @PKGx INTEGER,
                                                @EntityCIDx VARCHAR(12)
                                                        )
AS
BEGIN
/*
*******************************************************************************************
Name: usp_PSPriceTransportAll

Description: This is the main process for calculating the transport costs
for all transport entries within the gas inventory database.  These are
identified in the gas inventory database as PriceType=3 purchase and sale
entries (DBCR=0 or 1).

The recalculation of costs will only be allowed to occur when the gas month
status has been set to the appropriate month

Inputs:

GasMonthx - Gas Month to calculate
WhichPricex - 0=Nominations, 1=Actualizations
PKGx - either 0 for all or a specific package (deal) number
EntityCIDx - owning company id

History:

06/30/1999 JAMIE Orignal Creation.

03/22/2000 JAMIE Modified to move the Divie process to the main module. In addition,
modified to handle the new routing table (LegDetail) and build routing records
based on the routing rules within this table.

05/24/2000 JAMIE Modified to be aware of entity and product types and services.  In
addition, modifications made to calculate based on new routing structure...
*******************************************************************************************
*/
/*
```

```
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @zPackage INTEGER
DECLARE @zRecTID INTEGER
DECLARE @zDelTID INTEGER
DECLARE @zVolume DECIMAL(19,2)
DECLARE @zAmount DECIMAL(19,2)
DECLARE @zRate DECIMAL(19,8)
DECLARE @zLastDay DATETIME

DECLARE @yTID INTEGER

DECLARE @yGasDay DATETIME
DECLARE @yDelMID INTEGER
DECLARE @yRecMID INTEGER
DECLARE @yLID INTEGER
DECLARE @yReceipt DECIMAL(19,2)
DECLARE @yFuelOrOther DECIMAL(19,2)
DECLARE @yDelivered DECIMAL(19,2)
DECLARE @yTransportationRate DECIMAL(19,8)
DECLARE @yGatheringRate DECIMAL(19,8)
DECLARE @yFuelPercent DECIMAL(19,8)
DECLARE @yPlantVolReduction DECIMAL(19,8)
DECLARE @yKID INTEGER
DECLARE @yRMeterPipe VARCHAR(12)
DECLARE @yRMeterMeter VARCHAR(14)
DECLARE @yDMeterPipe VARCHAR(12)
DECLARE @yDMeterMeter VARCHAR(14)
DECLARE @yCID VARCHAR(12)
DECLARE @yKProductID INTEGER
DECLARE @yKServiceID INTEGER
DECLARE @yPurchasePKG INTEGER
/*
*******************************************
* First,intialize any existing volumes for
* this month on the gas inventory table
* to a zero.  In addition, set the
* appropriate volume amounts and price
* amounts on the 'Engine' table to zeros.
*******************************************
*/
EXECUTE usp_fLastDay @GasMonthx,@zLastDay OUTPUT
SELECT @zMessage = 'PSPriceTranportAll, Initializing Gas Inventory and Engine Information....'
EXECUTE usp_Message @zMessage
DECLARE GasInvCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                GasInv.TID
                FROM
                        GasInv,
                        K
                WHERE
                        GasInv.GasMonth=@GasMonthx AND
                        GasInv.PriceType=3 AND
                        K.KID=GasInv.KID AND
                        K.EntityCID=@EntityCIDx
OPEN GasInvCursor
FETCH NEXT FROM GasInvCursor INTO @yTID
BEGIN TRANSACTION
WHILE @@FETCH_STATUS = 0
        BEGIN
                IF @WhichPricex=0
                        BEGIN
                                UPDATE
                                        GasInvD
                                        SET
                                                Nom=0,
                                                    EstAct=0
                                            WHERE
                                                    TID=@yTID AND
                                                    GasDay BETWEEN @GasMonthx AND
@zLastDay
                                    UPDATE
                                            Engine
                                            SET
                                                    Volume=0,
                                                    Amount=0,
                                                    PriceOrRateNom=0
                                            WHERE
                                                    TID=@yTID
                            END
                    IF @WhichPricex=1
                            BEGIN
                                    UPDATE
                                            GasInvD
                                            SET
                                                    PipelineActuals=0
                                            WHERE
                                                    TID=@yTID AND
                                                    GasDay BETWEEN @GasMonthx AND
@zLastDay
                                    UPDATE
                                            Engine
                                            SET
                                                    VolumeAct=0,
                                                    AmountAct=0,
                                                    PriceOrRateAct=0
                                            WHERE
                                                    TID=@yTID
                            END
                    FETCH NEXT FROM GasInvCursor INTO @yTID
            END
SELECT @zMessage = 'PSPriceTranportAll, Finished initializing Gas Inventory and Engine Information....'
EXECUTE usp_Message @zMessage
COMMIT WORK
CLOSE GasInvCursor
DEALLOCATE GasInvCursor
/*
*********************************************
* Now loop through each of leg detail
* records for the month for this entity
* and determine appropriate transportation
* rates.
*
* Gas Inventory (PriceType=3) records will
* be created (along with package if needed).
*
* Engine records will also be created.
*********************************************
*/
SELECT @zMessage = 'PSPriceTranportAll, Analyzing Routing (legdetail) cursor....'
EXECUTE usp_Message @zMessage
DECLARE LegDetailCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
            SELECT
                    LD.GasDay,
                    LD.DelMID,
                    LD.RecMID,
                    LD.LID,
                    LD.Receipt,
                    LD.FuelOrOther,
                    LD.Delivered,
                    LD.TransportationRate,
                    LD.GatheringRate,
                    LD.FuelPercent,
                    LD.PlantVolReduction,
                    LD.PurchasePKG,
                    RMeter.PipeField,
                    RMeter.Meter,
                    DMeter.PipeField,
                    DMeter.Meter,
                    LegRef.KID
                    FROM
                            LegDetail AS LD,
                            LegRef,
                            Meter AS RMeter,
                            Meter AS DMeter
                    WHERE
                            LegRef.LID=LD.LID AND
                            RMeter.MID=LD.RecMID AND
                            DMeter.MID=LD.DelMID AND
                            LD.PurchasePointTID IN (SELECT TID FROM GasInv,Package,K WHERE
Package.PKG=GasInv.PKG AND K.KID = Package.KID AND
                                                                    GasInv.GasMonth=@GasMonthx
and GasInv.DBCR=0 and GasInv.PriceType=1 and K.EntityCID=@EntityCIDx) AND
                            LD.GasMonth=@GasMonthx AND
                            LD.GasDay>=@GasMonthx AND
                            LD.GasDay<=@zLastDay AND
                            LD.NomOrActuals=@WhichPricex AND
                            LD.LID<>0 AND
                            (LD.TransportationRate<>0 OR LD.GatheringRate<>0 OR LD.FuelPercent<>0 OR
LD.PlantVolReduction<>0)
                    ORDER BY
                            LegRef.LID
OPEN LegDetailCursor
FETCH NEXT FROM LegDetailCursor INTO @yGasDay,@yDelMID,@yRecMID,@yLID,@yReceipt,@yFuelOrOther,

          @yDelivered,@yTransportationRate,@yGatheringRate,@yFuelPercent,@yPlantVolReduction,@yPurchaseP
KG,

          @yRMeterPipe,@yRMeterMeter,@yDMeterPipe,@yDMeterMeter,@yKID
WHILE @@FETCH_STATUS = 0
          BEGIN
                    BEGIN TRANSACTION
                    /*
                    *******************************************
                    * First check to see if a transportation
                    * package has been setup for this
                    * contract/company... If not, then set
                    * it up... A commit is immediately
                    * performed here in order to 'preserve'
                    * the package information (and its
                    * associated ID).
                    *******************************************
                    */
                    SELECT @yKProductID=KProductID,@yKServiceID=KServiceID FROM Package where
PKG=@yPurchasePKG
                    SELECT @yCID=CID FROM K WHERE KID = @yKID
                    SELECT @zPackage=ISNULL((SELECT PKG FROM Package WHERE KID=@yKID AND
PackageGasMonth=@GasMonthx AND
                                                                    KProductID=@yKProductID AND
KServiceID=@yKServiceID),'')
                    IF (@zPackage='') OR (@zPackage IS NULL)
                            BEGIN
                                    SELECT @zPackage=(SELECT max(PKG) FROM package) + 1
                                    INSERT
                                            INTO
                                                    Package

          (PKG,StartDate,EndDate,Description,Package_Create,KID,CID,PackageGasMonth,PackageStatus,Package
_CreateBy,

          LastUpdateBy,LastUpdateDate,KProductID,KServiceID)
                                            VALUES

          (@zPackage,@GasMonthx,@zLastDay,'TRANSPORT
DEAL',getdate(),@yKID,@yCID,@GasMonthx,'Created',user_name(),
          user_name(),getdate(),@yKProductID,@yKServiceID)
                    END
               /*
               ******************************************
               * At this point we know that a package
               * has been created AND we have the
               * package identifier.  Now build the
               * GasInv records IF they do not already
               * exist for this package.  By adding a
               * new inventory item the daily (GasInvD)
               * records are automatically created for
               * each day of the month.
               ******************************************
               */
               SELECT @zRecTID=ISNULL((SELECT TID FROM GasInv WHERE GasMonth=@GasMonthx
AND
                                        PKG=@zPackage AND PriceType=3 AND
KID=@yKID AND PipeField=@yRMeterPipe AND
                                        Meter=@yRMeterMeter AND PID=@yLID
AND DBCR=0),0)
               IF @zRecTID=0
                    BEGIN
                         INSERT
                              INTO
                                   GasInv

          (GasMonth,CID,PipeField,Meter,DBCR,KID,PID,PKG,Stat,PriceType,GasInv_UT,

          Nom,EstAct,GasInv_UU,GasInv_MID,PipelineActuals)
                              VALUES

          (@GasMonthx,@yCID,@yRMeterPipe,@yRMeterMeter,0,@yKID,@yLID,@zPackage,1,3,getdate(),
                                        0,0,user_name(),@yRecMID,0)
                         SELECT @zRecTID=ISNULL((SELECT TID FROM GasInv WHERE
GasMonth=@GasMonthx AND

          PKG=@zPackage AND PriceType=3 AND KID=@yKID AND PipeField=@yRMeterPipe AND

          Meter=@yRMeterMeter AND PID=@yLID AND DBCR=0),0)
                    END

               SELECT @zDelTID=ISNULL((SELECT TID FROM GasInv WHERE GasMonth=@GasMonthx
AND PKG=@zPackage AND
                                             PriceType=3 AND KID=@yKID
AND PipeField=@yDMeterPipe AND Meter=@yDMeterMeter AND
                                             PID=@yLID AND DBCR=1),0)
               IF @zDelTID=0
                    BEGIN
                         INSERT
                              INTO
                                   GasInv

          (GasMonth,CID,PipeField,Meter,DBCR,KID,PID,PKG,Stat,PriceType,GasInv_UT,

          Nom,EstAct,GasInv_UU,GasInv_MID,PipelineActuals)
                              VALUES

          (@GasMonthx,@yCID,@yDMeterPipe,@yDMeterMeter,1,@yKID,@yLID,@zPackage,1,3,getdate(),
                                        0,0,user_name(),@yDelMID,0)
                         SELECT @zDelTID=ISNULL((SELECT TID FROM GasInv WHERE
GasMonth=@GasMonthx AND PKG=@zPackage AND

          PriceType=3 AND KID=@yKID AND PipeField=@yDMeterPipe AND Meter=@yDMeterMeter AND

          PID=@yLID AND DBCR=1),0)
                    END
               /*
               ******************************************
               * At this point the gas package and gas
```

```
                        * inventory items have been determined
                        * (created if needed).  Now go and post
                        * the volume to the GasInvD table.
                        ******************************************
                        */
                        IF @WhichPricex=0
                                BEGIN
                                        UPDATE
                                                GasInvD
                                                SET
                                                        nom=(nom+@yReceipt)
                                                WHERE
                                                        TID=@zRecTID AND
                                                        GasDay=@yGasDay
                                        UPDATE
                                                GasInvD
                                                SET
                                                        nom=(nom+@yDelivered)
                                                WHERE
                                                        TID=@zDelTID AND
                                                        GasDay=@yGasDay
                                END
                        IF @WhichPricex=1
                                BEGIN
                                        UPDATE
                                                GasInvD
                                                SET
                                                        PipelineActuals=(PipelineActuals+@yReceipt)
                                                WHERE
                                                        TID=@zRecTID AND
                                                        GasDay=@yGasDay
                                        UPDATE
                                                GasInvD
                                                SET
                PipelineActuals=(PipelineActuals+@yDelivered)
                                                WHERE
                                                        TID=@zDelTID AND
                                                        GasDay=@yGasDay
                                END
                        /*
                        ******************************************
                        * Any transport costs here???
                        * (engine transaction ID is 3)
                        ******************************************
                        */
                        IF @yTransportationRate<>0
                                BEGIN
                                        SELECT @zRate=@yTransportationRate
                                        SELECT @zVolume=@yReceipt
                                        SELECT @zAmount=ROUND((@zRate*@zVolume),2)
                                        IF ISNULL((SELECT count(*) FROM Engine WHERE TID=@zRecTID
        AND Effective=@GasMonthx AND STID=3),0) < 1
                                                BEGIN
                                                IF @WhichPricex=0
                                                        BEGIN
                                                        INSERT
                                                                INTO
                Engine
                (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,Volume,Amount,PriceOrRateNo
        m)
                                                                VALUES
                (@zRecTID,@GasMonthx,3,0,@zPackage,1,getdate(),user_name(),@zVolume,@zAmount,@zRate)
                                                        END
                                                IF @WhichPricex=1
                                                        BEGIN
                                                        INSERT
```

```
                                                                    INTO
            Engine
            (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,VolumeAct,AmountAct,PriceOrR
ateAct)
                                                                    VALUES
            (@zRecTID,@GasMonthx,3,0,@zPackage,1,getdate(),user_name(),@zVolume,@zAmount,@zRate)
                                                        END
                                            END
                                    ELSE
                                            BEGIN
                                            IF @WhichPricex=0
                                                        BEGIN
                                                        UPDATE
                                                                    Engine
                                                                    SET
            Volume=(Volume+@zVolume),
            Amount=(Amount+@zAmount),
            PriceOrRateNom=ROUND(((Amount+@zAmount)/(Volume+@zVolume)),4)
                                                              WHERE
            TID=@zRecTID AND
            Effective=@GasMonthx AND
            STID=3
                                                        END
                                                  IF @WhichPricex=1
                                                        BEGIN
                                                        UPDATE
                                                                    Engine
                                                                    SET
            VolumeAct=(VolumeAct+@zVolume),
            AmountAct=(AmountAct+@zAmount),
            PriceOrRateAct=ROUND(((AmountAct+@zAmount)/(VolumeAct+@zVolume)),4)
                                                              WHERE
            TID=@zRecTID AND
            Effective=@GasMonthx AND
            STID=3
                                                        END
                                            END
                              END
                        /*
                        ******************************************
                        * Any gathering costs here???
                        * (engine transaction ID is 4)
                        ******************************************
                        */
                        IF @yGatheringRate<>0
                              BEGIN
                                    SELECT @zRate=@yGatheringRate
                                    SELECT @zVolume=@yReceipt
                                    SELECT @zAmount=ROUND((@zRate*@zVolume),2)
                                    IF ISNULL((SELECT count(*) FROM Engine WHERE TID=@zRecTID
AND Effective=@GasMonthx AND STID=4),0) < 1
                                            BEGIN
                                            IF @WhichPricex=0
                                                        BEGIN
                                                        INSERT
```

```
                                                                                INTO
                        Engine
                        (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,Volume,Amount,PriceOrRateNo
                  m)
                                                                                VALUES
                        (@zRecTID,@GasMonthx,4,0,@zPackage,1,getdate(),user_name(),@zVolume,@zAmount,@zRate)
                                                                  END
                                                    IF @WhichPricex=1
                                                                  BEGIN
                                                                  INSERT
                                                                                INTO
                        Engine
                        (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,VolumeAct,AmountAct,PriceOrR
                  ateAct)
                                                                                VALUES
                        (@zRecTID,@GasMonthx,4,0,@zPackage,1,getdate(),user_name(),@zVolume,@zAmount,@zRate)
                                                                  END
                                                    END
                                              ELSE
                                                    BEGIN
                                                    IF @WhichPricex=0
                                                                  BEGIN
                                                                  UPDATE
                                                                                Engine
                                                                                SET
                        Volume=(Volume+@zVolume),
                        Amount=(Amount+@zAmount),
                        PriceOrRateNom=ROUND(((Amount+@zAmount)/(Volume+@zVolume)),4)
                                                                         WHERE
                        TID=@zRecTID AND
                        Effective=@GasMonthx AND
                        STID=4
                                                                  END
                                                          IF @WhichPricex=1
                                                                  BEGIN
                                                                  UPDATE
                                                                                Engine
                                                                                SET
                        VolumeAct=(VolumeAct+@zVolume),
                        AmountAct=(AmountAct+@zAmount),
                        PriceOrRateAct=ROUND(((AmountAct+@zAmount)/(VolumeAct+@zVolume)),4)
                                                                         WHERE
                        TID=@zRecTID AND
                        Effective=@GasMonthx AND
                        STID=4
                                                                  END
                                                    END
                                      END
                              /*
                              *******************************************
                              * Any fuel costs??
                              * (engine transaction ID is 5)
```

```
                    *******************************************
                    */
                    IF @yFuelPercent<>0
                            BEGIN
                                    SELECT @zRate=@yFuelPercent
                                    SELECT @zVolume=@yReceipt*@zRate
                                    IF ISNULL((SELECT count(*) FROM Engine WHERE TID=@zRecTID
AND Effective=@GasMonthx AND STID=5),0) < 1
                                            BEGIN
                                            IF @WhichPricex=0
                                                            BEGIN
                                                            INSERT
                                                                            INTO
        Engine
        (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,Volume,Amount,PriceOrRateNo
m)
                                                                            VALUES
        (@zRecTID,@GasMonthx,5,0,@zPackage,1,getdate(),user_name(),@zVolume,0,@zRate)
                                                            END
                                            IF @WhichPricex=1
                                                            BEGIN
                                                            INSERT
                                                                            INTO
        Engine
        (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,VolumeAct,AmountAct,PriceOrR
ateAct)
                                                                            VALUES
        (@zRecTID,@GasMonthx,5,0,@zPackage,1,getdate(),user_name(),@zVolume,0,@zRate)
                                                            END
                                            END
                                    ELSE
                                            BEGIN
                                            IF @WhichPricex=0
                                                            BEGIN
                                                            UPDATE
                                                                            Engine
                                                                            SET
        Volume=(Volume+@zVolume)
                                                                    WHERE
        TID=@zRecTID AND
        Effective=@GasMonthx AND
        STID=5
                                                            END
                                                    IF @WhichPricex=1
                                                            BEGIN
                                                            UPDATE
                                                                            Engine
                                                                            SET
        VolumeAct=(VolumeAct+@zVolume)
                                                                    WHERE
        TID=@zRecTID AND
        Effective=@GasMonthx AND
        STID=5
                                                            END
                                            END
                            END
                    /*
                    ********************************************
                    * Any pvr??
                    * (engine transaction ID is 6)
                    ********************************************
                    */
                    IF @yPlantVolReduction<>0
                                BEGIN
                                            SELECT @zRate=@yPlantVolReduction
                                            SELECT @zVolume=@yReceipt*@zRate
                                            IF ISNULL((SELECT count(*) FROM Engine WHERE TID=@zRecTID
AND Effective=@GasMonthx AND STID=6),0) < 1
                                                        BEGIN
                                                        IF @WhichPricex=0
                                                                                BEGIN
                                                                                INSERT
                                                                                                    INTO

          Engine

          (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,Volume,Amount,PriceOrRateNo
m)
                                                                                                    VALUES

          (@zRecTID,@GasMonthx,6,0,@zPackage,1,getdate(),user_name(),@zVolume,0,@zRate)
                                                                                END
                                                        IF @WhichPricex=1
                                                                                BEGIN
                                                                                INSERT
                                                                                                    INTO

          Engine

          (TID,Effective,STID,VolLevel,VolGroup,MMBtuMCF,Engine_UT,Engine_UU,VolumeAct,AmountAct,PriceOrR
ateAct)
                                                                                                    VALUES

          (@zRecTID,@GasMonthx,6,0,@zPackage,1,getdate(),user_name(),@zVolume,0,@zRate)
                                                                                END
                                                        END
                                            ELSE
                                                        BEGIN
                                                        IF @WhichPricex=0
                                                                                BEGIN
                                                                                UPDATE
                                                                                                    Engine
                                                                                                    SET

          Volume=(Volume+@zVolume)
                                                                                          WHERE

          TID=@zRecTID AND

          Effective=@GasMonthx AND

          STID=6
                                                                                END
                                                                    IF @WhichPricex=1
                                                                                BEGIN
                                                                                UPDATE
                                                                                                    Engine
                                                                                                    SET

          VolumeAct=(VolumeAct+@zVolume)
                                                                                          WHERE

          TID=@zRecTID AND

          Effective=@GasMonthx AND
```

```
                STID=6
                                                        END
                                                END
                                        END
                                COMMIT WORK
                                FETCH NEXT FROM LegDetailCursor INTO
@yGasDay,@yDelMID,@yRecMID,@yLID,@yReceipt,@yFuelOrOther,

        @yDelivered,@yTransportationRate,@yGatheringRate,@yFuelPercent,@yPlantVolReduction,@yPurchaseP
KG,

        @yRMeterPipe,@yRMeterMeter,@yDMeterPipe,@yDMeterMeter,@yKID
        END
CLOSE LegDetailCursor
DEALLOCATE LegDetailCursor
SELECT @zMessage = 'PSPriceTranportAll, Finished....'
EXECUTE usp_Message @zMessage
END









GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceWASPCalc(
                                @GasMonthx DATETIME,
                                @WhichPricex INTEGER,
                                @EntityCIDx VARCHAR(12)
                                                )
AS
BEGIN
/*
******************************************************************************************
Name: usp_PSPriceWaspCalc

Description: This is the main process for calculating the WASP price information for
a particular gas month and type of price (nom's or pipeline actuals). The end result
of this process is to post updated price amounts within the engine. The WASP calculation
has also been modified to perform the calculations pooled by entity (passed to this
routine), within entity by product (Oil/Gas/Liguids) and service (marketing/passthrough/etc.).

Inputs:

GasMonthx (Gas Month to calculate),
WhichPricex (0=Nominations, 1=Actualizations)
EntityCIDx (which company is being calculated (owner company))

History:

06/22/99 JAMIE Original creation

07/22/99 JAMIE Include 3rd party deals within the
calcualtion process. They WILL NOT BE included within the WASP calculations
and will be treated the same as "Dedicated" (sanctioned sales) deals. This
will ensure they are not affecting any other pricing component.

05/01/00 JAMIE Modifications to utilize the new routing structure as part of the
calculation. A check is made to see if any 'routing' entries are made to the new
structures (for the month). If so, then this routine will invoke the new routines.
```

```
Otherwise, the old routines are invoked.

05/24/2000 JAMIE Modifications to add the EntityCIDx component to the calculation (passed
to this routine by the calling program). In addition, modifications were made to calculate
all WASP pricing within each unique product and service.

08/25/2000 JAMIE Modified to remove all of the old routing routines.

*******************************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @yKProductID INTEGER
DECLARE @yKProductName VARCHAR(50)

DECLARE @yKServiceID INTEGER
DECLARE @yKServiceName VARCHAR(50)

DECLARE ProductTypesCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                ProcessingCodeID,
                ShortDescription
                FROM
                        SEProcessingCodes WHERE CodeType='CONTRPRODS'
                ORDER BY
                        ProcessingCodeID

SELECT @zMessage = 'PSPriceWASPCalc, Running for Entity '+@EntityCIDx+'...'
EXECUTE usp_Message @zMessage
/*
*******************************************
* Outermost loop is on product type...
*******************************************
*/
OPEN ProductTypesCursor
FETCH NEXT FROM ProductTypesCursor INTO @yKProductID,@yKProductName
WHILE @@FETCH_STATUS = 0
        BEGIN
                SELECT @zMessage = 'PSPriceWASPCalc, Running for Product '+@yKProductName+'...'
                EXECUTE usp_Message @zMessage
                /*
                *******************************************
                * Next loop is on service type...
                *******************************************
                */
                DECLARE ServiceTypesCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                        SELECT
                                ProcessingCodeID,
                                ShortDescription
                                FROM
                                        SEProcessingCodes WHERE CodeType='CONTRSRVS'
                                ORDER BY
                                        ProcessingCodeID
                OPEN ServiceTypesCursor
                FETCH NEXT FROM ServiceTypesCursor INTO @yKServiceID,@yKServiceName
                WHILE @@FETCH_STATUS = 0
                        BEGIN
                                BEGIN TRANSACTION
                                SELECT @zMessage = 'PSPriceWASPCalc, Running for Service
'+@yKServiceName+'..'
                                EXECUTE usp_Message @zMessage
                                /*
                                *******************************************
                                * Now populate the waspresolvedrouting
                                * tables with all sales and transport
```

```
                                        * totals that were linked to purchases
                                        * within the route process.
                                        ********************************************
                                        */
                                        EXECUTE usp_PSPriceWASPCalcSalesN
@GasMonthx,@WhichPricex,@EntityCIDx,@yKProductID,@yKServiceID
                                        COMMIT WORK
                                        FETCH NEXT FROM ServiceTypesCursor INTO
@yKServiceID,@yKServiceName
                                  END
                        CLOSE ServiceTypesCursor
                        DEALLOCATE ServiceTypesCursor
                        FETCH NEXT FROM ProductTypesCursor INTO @yKProductID,@yKProductName
            END
CLOSE ProductTypesCursor
DEALLOCATE ProductTypesCursor
/*
********************************************
* Finished.  A later routine will take
* the well prices to the actual engine
* table (PSPriceAll for Purchases).  A
* commit takes place right here just to
* make sure we limit our recovery window
* if problems later..  Also, don't want
* to hold locks for an extended amount
* of time.
********************************************
*/
SELECT @zMessage = 'PSPriceWASPCalc, Finished with Entity '+@EntityCIDx+'...'
EXECUTE usp_Message @zMessage
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceWASPCalcResolveDriver(
                                                                  @GasMonthx DATETIME,
                                                                  @WhichPricex INTEGER,
                                                                  @EntityCIDx VARCHAR(12),
                                                                  @IncludeInWaspx VARCHAR(10)
                                                                  )
AS
BEGIN
/*
******************************************************************************************
Name: usp_PSPriceWaspCalcResolveDriver

Description: This is the main process that controls the 'walking back' (resolving)
of sales amounts back to their respective purchase deals.

Inputs:

GasMonthx (Gas Month to calculate),
WhichPricex (0=Nominations, 1=Actualizations)
EntityCIDx (which company is being calculated (owner company))
IncludeInWaspx ('Common','None' or 'Dedicated')

History:

07/28/2000 JAMIE Original creation

******************************************************************************************
```

```
*/
/*
********************************************
* Declare all variables and cursors
* that are needed by this process.
********************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @yKProductID INTEGER
DECLARE @yKProductName VARCHAR(50)

DECLARE @yKServiceID INTEGER
DECLARE @yKServiceName VARCHAR(50)

DECLARE ProductTypesCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                ProcessingCodeID,
                ShortDescription
                FROM
                        SEProcessingCodes
                WHERE
                        CodeType='CONTRPRODS'
                ORDER BY
                        ProcessingCodeID

SELECT @zMessage = 'PSPriceWASPCalcResloveDriver, Running for Entity '+@EntityCIDx+',Pool
'+@IncludeInWaspx+'...'
EXECUTE usp_Message @zMessage
/*
********************************************
* Outermost loop is on product type...
********************************************
*/
OPEN ProductTypesCursor
FETCH NEXT FROM ProductTypesCursor INTO @yKProductID,@yKProductName
WHILE @@FETCH_STATUS = 0
        BEGIN
                SELECT @zMessage = 'PSPriceWASPCalcResloveDriver, Running for Product
'+@yKProductName+'...'
                EXECUTE usp_Message @zMessage
                /*
                ********************************************
                * Next loop is on service type...
                ********************************************
                */
                DECLARE ServiceTypesCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                        SELECT
                                ProcessingCodeID,
                                ShortDescription
                                FROM
                                        SEProcessingCodes
                                WHERE
                                        CodeType='CONTRSRVS'
                                ORDER BY
                                        ProcessingCodeID
                OPEN ServiceTypesCursor
                FETCH NEXT FROM ServiceTypesCursor INTO @yKServiceID,@yKServiceName
                WHILE @@FETCH_STATUS = 0
                        BEGIN
                                BEGIN TRANSACTION
                                SELECT @zMessage = 'PSPriceWASPCalcResloveDriver, Running for
Service '+@yKServiceName+' ..'
                                EXECUTE usp_Message @zMessage
                                EXECUTE usp_PSPriceWASPCalcResolveN
@GasMonthx,@WhichPricex,@EntityCIDx,@yKProductID,@yKServiceID,@IncludeInWaspx
                                COMMIT WORK
                                FETCH NEXT FROM ServiceTypesCursor INTO
@yKServiceID,@yKServiceName
                        END
                CLOSE ServiceTypesCursor
```

```
                    DEALLOCATE ServiceTypesCursor
                    FETCH NEXT FROM ProductTypesCursor INTO @yKProductID,@yKProductName
            END
        CLOSE ProductTypesCursor
        DEALLOCATE ProductTypesCursor
        SELECT @zMessage = 'PSPriceWASPCalcResolveDriver, Finished with Entity '+@EntityCIDx+',Pool
        '+@IncludeInWaspx+'...'
        EXECUTE usp_Message @zMessage
        END




        GO
        SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
        GO

        SET QUOTED_IDENTIFIER  ON    SET ANSI_NULLS  ON
        GO

        CREATE PROCEDURE usp_PSPriceWASPCalcResolveN(
                                                          @GasMonthx DATETIME,
                                                          @WhichPricex INTEGER,
                                                          @EntityCIDx VARCHAR(12),
                                                          @KProductIDx INTEGER,
                                                          @KServiceIDx INTEGER,
                                                          @IncludeInWaspx VARCHAR(10)
                                                          )
        AS
        BEGIN
        /*
        ******************************************************************************************
        Name: usp_PSPriceWASPCalcResolveN

        Description: This particular stored procedure is responsible for looping through and
        chasing all volumes back from purchase points back to the respective meter locations
        that originally contained the purchase volumes.

        History:

        05/01/2000 JAMIE Original Creation.

        05/24/2000 JAMIE Modified to include the entity, product and service.

        07/28/2000 JAMIE Modified to include the IncludeInWaspx parameter so that
        the calculations can be run in a specified WASP order...

        08/17/2000 JAMIE Removed the call to PSWASPCalcPostPurchaseN. This
        was done based on all wasp calculation entries being setup in the
        WASPResolvedRouting table.

        ******************************************************************************************
        */
        /*
         ********************************************
         * Declare all variables and cursors
         * that are needed by this process.
         ********************************************
         */
        DECLARE @zMessage VARCHAR(254)

        SELECT @zMessage = 'PSPriceWASPCalcResolveN Has Started for pool '+@IncludeInWaspx+'...'
        EXECUTE usp_Message @zMessage
        /*
        ********************************************
        * Now invoke the routine that will chase
        * the volumes, prices and amounts back to
        * the purchase points.
        ********************************************
```

```
*/
SELECT @zMessage = 'PSPriceWASPCalcResolveN, Tracing back all gas (resolving sales)...'
EXECUTE usp_Message @zMessage
EXECUTE usp_PSPriceWASPCalcResolveSalesN
@GasMonthx,@WhichPricex,@EntityCIDx,@KProductIDx,@KServiceIDx,@IncludeInWaspx
/*
*******************************************
* Time to leave...
*******************************************
*/
SELECT @zMessage =  'PSPriceWASPCalcResolveN Has Completed for Pool '+@IncludeInWaspx+'...'
EXECUTE usp_Message @zMessage
END


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceWASPCalcResolveSalesN(
                                                                            @GasMonthx
DATETIME,
                                                                            @WhichPricex
INTEGER,
                                                                            @EntityCIDx
VARCHAR(12),
                                                                            @KProductIDx
INTEGER,
                                                                            @KServiceIDx
INTEGER,
                                                                            @IncludeInWaspx
VARCHAR(10)
                                                                            )
AS
BEGIN
/*
*****************************************************************************************
Name: usp_PSPriceWASPCalcResolveSales

Description: This particular stored procedure will loop through (iteratively) all of
the sales meter records within the WASPResolvedRouting table (type 'S' records) and
distribute their respective volumes, amounts and prices back to the purchase points
(wieghted).

All volumes should match here since the routing process routes purchase deals directly
to sales deals AND the WASPResolvedRouting table was built on explicit volumes and
links found in the LegDetail (main routing) table.

Inputs:

GasMonthx - Gas Month
WhichPricex - 0=Nominations, 1=Actuals
EntityCIDx - owning company
KProductIDx - product id (oil, gas, liquids, etc.)
KServiceIDx - service id (marketing, passthrough, etc.)
IncludeInWaspx - ('Common' or 'None' or 'Dedicated')

History:

05/01/2000 JAMIE Original Creation.

07/20/2000 JAMIE Modified in order to capture and save resolved total amounts
along with the resolved volume amounts.  This was required in order to correct a
calculation problem.

07/28/2000 JAMIE Modified to take into consideration which WASP pool is currently
being resolved.
```

```
12/05/2000 JAMIE Modified to ensure that the receipt amount will not be exceeded
when determining the volume to use. This situation only arose when certain
unresolved records were ordered a certain way (during the resolution ritual).
Confusing, I know, but that is the best I can do... The field zTempLeft contains
this informaion

*****************************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zTempLeft DECIMAL(19,2)
DECLARE @zRound INTEGER
DECLARE @zMessage VARCHAR(254)
DECLARE @zAnyUpdates VARCHAR(1)
DECLARE @zResolvedReceipt DECIMAL(19,2)
DECLARE @zResolvedReceiptAmt DECIMAL(19,2)
DECLARE @zResolvedDelivered DECIMAL(19,2)
DECLARE @zResolvedDeliveredAmt DECIMAL(19,2)
DECLARE @zReceiptLeft DECIMAL(19,2)
DECLARE @zReceiptAmtLeft DECIMAL(19,2)
DECLARE @zPercentToApply DECIMAL(19,6)
DECLARE @zSumDelivered DECIMAL(19,2)
DECLARE @zPercentReceipt DECIMAL(19,6)
DECLARE @zUseVolume DECIMAL(19,2)
DECLARE @zUseAmount DECIMAL(19,2)
DECLARE @zAmount DECIMAL(19,2)
DECLARE @zNewAmount DECIMAL(19,2)
DECLARE @zNewPrice DECIMAL(19,6)
DECLARE @zTempVolume DECIMAL(19,2)
DECLARE @zTempAmount DECIMAL(19,2)
DECLARE @zVolumeDispersed DECIMAL(19,2)
DECLARE @zAmountDispersed DECIMAL(19,2)
DECLARE @zDifference DECIMAL(19,2)
DECLARE @zResolvedIndicator VARCHAR(1)
DECLARE @zLinkUpdate VARCHAR(1)
DECLARE @zDeliveredLeft DECIMAL(19,2)

DECLARE @yDelMID INTEGER
DECLARE @yRecMID INTEGER
DECLARE @yReceipt DECIMAL(19,2)
DECLARE @yFuelOrOther DECIMAL(19,2)
DECLARE @yDelivered DECIMAL(19,2)
DECLARE @yTransportAmount DECIMAL(19,2)
DECLARE @yGatheringAmount DECIMAL(19,2)
DECLARE @yAmount DECIMAL(19,2)
DECLARE @yDedicatedPurchasePKG INTEGER
DECLARE @yPrice DECIMAL(19,6)
DECLARE @yResolvedReceipt DECIMAL(19,2)
DECLARE @yIncludeInWasp VARCHAR(10)
DECLARE @yResolvedDelivered DECIMAL(19,2)
DECLARE @yResolvedID INTEGER
DECLARE @yResolvedReceiptAmt DECIMAL(19,2)
DECLARE @yResolvedDeliveredAmt DECIMAL(19,2)

DECLARE @lDelMID INTEGER
DECLARE @lRecMID INTEGER
DECLARE @lReceipt DECIMAL(19,2)
DECLARE @lFuelOrOther DECIMAL(19,2)
DECLARE @lDelivered DECIMAL(19,2)
DECLARE @lTransportAmount DECIMAL(19,2)
DECLARE @lGatheringAmount DECIMAL(19,2)
DECLARE @lAmount DECIMAL(19,2)
DECLARE @lDedicatedPurchasePKG INTEGER
DECLARE @lPrice DECIMAL(15,6)
DECLARE @lResolvedReceipt DECIMAL(19,2)
DECLARE @lIncludeInWasp VARCHAR(10)
DECLARE @lResolvedDelivered DECIMAL(19,2)
DECLARE @lResolvedID INTEGER
DECLARE @lResolvedReceiptAmt DECIMAL(19,2)
DECLARE @lResolvedDeliveredAmt DECIMAL(19,2)

/*
****************************************
* This loop will iterate until no more
* gas can be distributed to various
* sales meters within the
* WaspResolvedRouting table.
****************************************
*/
SELECT @zRound = ISNULL((SELECT TypeLimit FROM SEProcessingCodes WHERE ProcessingCodeID =
@KProductIDx),0)
SELECT @zMessage = 'PSPriceWASPCalcResolveSalesN, starting iterative process...'
EXECUTE usp_Message @zMessage
SalesMeterIterationLoop:
        BEGIN
                SELECT @zAnyUpdates='N'
                DECLARE WASPResolvedSalesCursor CURSOR LOCAL DYNAMIC FORWARD_ONLY FOR
                        SELECT
                                DelMID,
                                RecMID,
                                Receipt,
                                FuelOrOther,
                                Delivered,
                                TransportAmount,
                                GatheringAmount,
                                Amount,
                                DedicatedPurchasePKG,
                                Price,
                                ResolvedReceipt,
                                IncludeInWasp,
                                ResolvedDelivered,
                                ResolvedID,
                                ResolvedReceiptAmt,
                                ResolvedDeliveredAmt
                                FROM
                                        WASPResolvedRouting
                                WHERE
                                        (GasMonth=@GasMonthx AND
                                        NomOrActual=@WhichPricex AND
                                        IncludeInWasp=@IncludeInWaspx AND
                                        ResolvedIndicator<>'Y' AND
                                        ResolvedReceipt<>Receipt AND
                                        ResolvedType<>'P' AND
                                        Amount<>0 AND
                                        Price<>0 AND
                                        Delivered<>0 AND
                                        EntityCID=@EntityCIDx AND
                                        KProductID=@KProductIDx AND
                                        KServiceID=@KServiceIDx)
                                ORDER BY
                                        IncludeInWasp,
                                        DedicatedPurchasePKG,
                                        DelMID
                OPEN WASPResolvedSalesCursor
                FETCH NEXT FROM WASPResolvedSalesCursor INTO @yDelMID,

        @yRecMID,@yReceipt,@yFuelOrOther,@yDelivered,@yTransportAmount,@yGatheringAmount,@yAmount
,@yDedicatedPurchasePKG,

        @yPrice,@yResolvedReceipt,@yIncludeInWasp,@yResolvedDelivered,@yResolvedID,

        @yResolvedReceiptAmt,@yResolvedDeliveredAmt
                WHILE @@FETCH_STATUS = 0
                        BEGIN
                                /*
                        ****************************************
                        * Loop through each of the legs that
                        * have the delivery meter the same as
                        * the receipt meter for the given
                        * month and class...
                        ****************************************
                        */
                        SELECT @zVolumeDispersed=0
                        SELECT @zAmountDispersed=0
                        SELECT @zLinkUpdate='N'
                        DECLARE WASPResolvedLinkCursor CURSOR LOCAL DYNAMIC
FORWARD_ONLY FOR
                                SELECT
                                        DelMID,
                                        RecMID,
                                        Receipt,
                                        FuelOrOther,
                                        Delivered,
                                        TransportAmount,
                                        GatheringAmount,
                                        Amount,
                                        DedicatedPurchasePKG,
                                        Price,
                                        ResolvedReceipt,
                                        IncludeInWasp,
                                        ResolvedDelivered,
                                        ResolvedID,
                                        ResolvedReceiptAmt,
                                        ResolvedDeliveredAmt
                                        FROM
                                                WASPResolvedRouting
                                        WHERE
                                                (GasMonth=@GasMonthx AND
                                                NomOrActual=@WhichPricex
AND
                                                IncludeInWasp=@yIncludeInWasp
AND
        DedicatedPurchasePKG=@yDedicatedPurchasePKG AND
                                                DelMID=@yRecMID AND
                                                ResolvedID<>@yResolvedID
AND
                                                EntityCID=@EntityCIDx AND
                                                KProductID=@KProductIDx AND
                                                KServiceID=@KServiceIDx AND
                                                ResolvedType<>'S' AND
                                                ResolvedDelivered<Delivered)
                        OPEN WASPResolvedLinkCursor
                FETCH NEXT FROM WASPResolvedLinkCursor INTO @lDelMID,

        @lRecMID,@lReceipt,@lFuelOrOther,@lDelivered,@lTransportAmount,@lGatheringAmount,@lAmount,@l
DedicatedPurchasePKG,

        @lPrice,@lResolvedReceipt,@lIncludeInWasp,@lResolvedDelivered,@lResolvedID,

        @lResolvedReceiptAmt,@lResolvedDeliveredAmt
                        WHILE @@FETCH_STATUS = 0
                                BEGIN
                                /*
                                ****************************************
                                * Determine the total volume of gas
                                * where this gas came from (based on
                                * delivery meterid being equal to
                                * the receipt meter id and all WASP
                                * pool and dedicated purchase package
                                * information being identical).
                                *
                                * The zUseVolume field contains the
                                * amount of volume from the delivery
                                * meter to apply backward.
```

```
*
* The zUseAmount field contains the
* dollar amount from the delivery meter
* that should be applied backward.
*
* The zPercentToApply field contains the
* volume weighted percentage to use.
****************************************
*/
SELECT @zResolvedReceipt=@yResolvedReceipt
SELECT
@zResolvedReceiptAmt=@yResolvedReceiptAmt
SELECT @zPercentReceipt=1

/* Determine total receipt volume available to apply*/
/* This is based on percentage of delivered that may have*/
/* already been applied. In addition, determine the*/
/* amount that is available...*/

IF (@yDelivered<>0) AND (@yResolvedDelivered<>0)
AND (@yDelivered>@yResolvedDelivered)
        BEGIN
        SELECT
@zPercentReceipt=(@yResolvedDelivered/@yDelivered)
        END

    /* Incorporated this logic to ensure that no
more than */
    /* the original receipt can be sent back to
previous */
    /* meter... 12/05/2000 */

SELECT
@zReceiptLeft=ROUND((@yReceipt*@zPercentReceipt),@zRound)
    SELECT @zTempLeft=(@yReceipt -
@yResolvedReceipt)
    SELECT @zTempLeft=Round((@zTempLeft *
@zPercentReceipt),@zRound);
    IF @zTempLeft < @zReceiptLeft
        BEGIN
            SELECT
@zReceiptLeft=@zTempLeft
        END
SELECT @zReceiptAmtLeft=ROUND((@yAmount-
@yResolvedReceiptAmt),2)

/* Determine percentage of the volumes and amounts to
apply...  and RecMID<>DelMID        */

SELECT @zPercentToApply=1
SELECT @zSumDelivered=ISNULL((SELECT
SUM(Delivered) FROM WASPResolvedRouting
                                                WHERE
GasMonth=@GasMonthx AND NomOrActual=@WhichPricex AND IncludeInWasp=@yIncludeInWasp AND
        DedicatedPurchasePKG=@yDedicatedPurchasePKG AND DelMid=@yRecMID AND ResolvedType<>'S'
AND
        EntityCID=@EntityCIDx AND KProductID=@KProductIDx AND KServiceID=@KServiceIDx),0)

IF (@zSumDelivered<>0) AND (@lDelivered<>0)
        BEGIN
        SELECT
@zPercentToApply=ROUND((@lDelivered/@zSumDelivered),6)
        END
ELSE
        BEGIN
        SELECT @zPercentToApply=0
        END
```

```
/* Calculate volume to apply backwards for this particular leg...*/
SELECT @zUseVolume=ROUND((@zReceiptLeft*@zPercentToApply),@zRound)
SELECT @zDeliveredLeft=@lDelivered-@lResolvedDelivered
IF @zUseVolume>@zDeliveredLeft
        BEGIN
        SELECT @zUseVolume=@zDeliveredLeft
        END
SELECT @zResolvedReceipt=@zResolvedReceipt+@zUseVolume
SELECT @zVolumeDispersed=@zVolumeDispersed+@zUseVolume

/* Calculate dollar amount to apply backwards for this particular leg...*/
SELECT @zUseAmount=ROUND((@zReceiptAmtLeft*@zPercentToApply),2)
SELECT @zResolvedReceiptAmt=@zResolvedReceiptAmt+@zUseAmount
SELECT @zAmountDispersed=@zAmountDispersed+@zUseAmount
/*
*****************************************
* Now update the meter feeding
* this delivery point with the
* information just posted...
*
* The amount is calculated based
* on the previous value plus
* the amount being posted from
* the delivery meter.  The
* price is derived based on
* receipt volume into the amount.
*
* Since we are not forcing the pipes
* to balance then calculate the price
* based solely on the volume resolved
* on delivery.
*****************************************
*/
IF (@zUseVolume>0) AND (@zUseAmount<>0)
    BEGIN
        SELECT @zResolvedDelivered=@lResolvedDelivered+@zUseVolume
        SELECT @zResolvedDeliveredAmt=@lResolvedDeliveredAmt+@zUseAmount
        SELECT @zNewAmount=ROUND((@lAmount+@zUseAmount),2)
        IF (@zResolvedDeliveredAmt<>0) AND (@lReceipt<>0)
                BEGIN
                SELECT @zNewPrice=ROUND((@zNewAmount/@lReceipt),4)
                END
        ELSE
                BEGIN
                SELECT @zNewPrice=0
                END
        UPDATE WASPResolvedRouting
                SET ResolvedIndicator='N',
                ResolvedDelivered=@zResolvedDelivered.

                ResolvedDeliveredAmt=@zResolvedDeliveredAmt,

                Amount=@zNewAmount,

                Price=@zNewPrice
                                                                        WHERE

                ResolvedID=@lResolvedID
                                                        SELECT @zAnyUpdates='Y'
                                                        SELECT @zLinkUpdate='Y'
                                                END
                                        FETCH NEXT FROM WASPResolvedLinkCursor INTO
@lDelMID,

                @lRecMID,@lReceipt,@lFuelOrOther,@lDelivered,@lTransportAmount,@lGatheringAmount,@lAmount,@l
DedicatedPurchasePKG,

                @lPrice,@lResolvedReceipt,@lIncludeInWasp,@lResolvedDelivered,@lResolvedID,

                @lResolvedReceiptAmt,@lResolvedDeliveredAmt
                                        END
                                CLOSE WASPResolvedLinkCursor
                                DEALLOCATE WASPResolvedLinkCursor
                                /*
                                **********************************
                                * After looping through all of the
                                * meters that can possible associate
                                * with this sale, go ahead and update
                                * the original sales meter information
                                * to reflect the total volume
                                * passed on to subsequent meters.
                                **********************************
                                */
                                IF @zLinkUpdate='Y'
                                        BEGIN
                                                UPDATE
                                                        WASPResolvedRouting
                                                        SET

                ResolvedReceipt=ResolvedReceipt+@zVolumeDispersed,

                ResolvedReceiptAmt=ResolvedReceiptAmt+@zAmountDispersed,
                                                        ResolvedIndicator='Y'
                                                WHERE

                ResolvedID=@yResolvedID
                                        END
                                FETCH NEXT FROM WASPResolvedSalesCursor INTO @yDelMID,

                @yRecMID,@yReceipt,@yFuelOrOther,@yDelivered,@yTransportAmount,@yGatheringAmount,@yAmount
,@yDedicatedPurchasePKG,

                @yPrice,@yResolvedReceipt,@yIncludeInWasp,@yResolvedDelivered,@yResolvedID,

                @yResolvedReceiptAmt,@yResolvedDeliveredAmt
                        END
                CLOSE WASPResolvedSalesCursor
                DEALLOCATE WASPResolvedSalesCursor
                /*
                ****************************************
                * If no more volume was chased backward
                * then get out of the iterative loop.
                * At this point all volumes have been
                * sent back to all meters and weighted
                * costs should be available at each.
                ****************************************
                */
```

```
                        IF @zAnyUpdates<>'N'
                                BEGIN
                                        GOTO SalesMeterIterationLoop
                                END
        END
END




GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceWASPCalcSalesN(
                                                @GasMonthx DATETIME,
                                                @WhichPricex INTEGER,
                                                @EntityCIDx VARCHAR(12),
                                                @KProductIDx INTEGER,
                                                @KServiceIDx INTEGER
                                                )
AS
BEGIN
/*
**********************************************************************************************
Name: usp_PSPriceWASPCalcSalesN

Description: This process will build all of the meters within the
WASPResolvedRouting table for all of the deals within the gas month. Only
those meters that had actual transport volume will be moved.  A
different routine will iterate through the volumes posted here in order
to calculate all of the prices.

Inputs:

GasMonthx - Gas Month
WhichPricex - 0=Nominations, 1=Actuals
EntityCIDx - Entity being calculated (owning company)
KProductIDx - Product type being calculated.
KServiceIDx - Service type being calculated.

History:

05/02/2000 JAMIE Original Creation.

05/24/2000  JAMIE  Modified to add the Entity, product and service types to be
parameters to this procedure.  This will ensure that gas, oil, etc amongst the
various types of companies (entities) being serviced do not get intermixed.

07/20/2000 JAMIE Modified in order to initialize new resolved amount fields for
all records that get added to the WASPResolvedRouting table.

08/18/2000 JAMIE Modified to go ahead and put the actual purchase point
items on the table to include them in the calculations.  At this point the
WASPResolvedRouting table will contain ALL entries (see 'Type' field on the
database). Purchase points thru Sales points.

10/03/2000 JAMIE Modified to incorporate the 'Other Cost' amount totals
into the Resolved table total calculation.
```

```
01/09/2000 JAMIE For consistency.  Modified the rounding (on the prices
to two decimal places (for all months previous to December 2000).

***********************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @zIncludeInWasp VARCHAR(10)
DECLARE @zVolume DECIMAL(19,2)
DECLARE @zType VARCHAR(1)
DECLARE @zPrice DECIMAL(19,6)
DECLARE @zAmount DECIMAL(19,2)
DECLARE @zOtherCostAmount DECIMAL(19,2)
DECLARE @zDedicatedPurchasePKG INTEGER
DECLARE @zGatheringAmount DECIMAL(19,2)
DECLARE @zTransportationAmount DECIMAL(15,2)
DECLARE @zAmountWithCosts DECIMAL(19,2)
DECLARE @zLastDay DATETIME
DECLARE @zPrevSalePKG INTEGER
DECLARE @zPrevSaleMID INTEGER

DECLARE @yPurchasePKG INTEGER
DECLARE @yRecMID INTEGER
DECLARE @yDelMID INTEGER
DECLARE @ySalesPKG INTEGER
DECLARE @yReceipt DECIMAL(19,2)
DECLARE @yLDIDPrev INTEGER
DECLARE @yGasDay DATETIME
DECLARE @yPurchasePointTID INTEGER
DECLARE @yStep INTEGER

DECLARE @xPriceOrRateNom DECIMAL(19,6)
DECLARE @xPriceOrRateAct DECIMAL(19,6)

DECLARE @qPurchasePKG INTEGER
DECLARE @qLID INTEGER
DECLARE @qRecMID INTEGER
DECLARE @qDelMID INTEGER
DECLARE @qReceipt DECIMAL(19,2)
DECLARE @qDelivered DECIMAL(19,2)
DECLARE @qFuelOrOther DECIMAL(19,2)
DECLARE @qTransport DECIMAL(19,2)
DECLARE @qGathering DECIMAL(19,2)


SELECT @zMessage = 'PSPriceWASPCalcSalesN Has Started...'
EXECUTE usp_Message @zMessage
/*
*******************************************
* Delete any pre-existing resolved entries
* that may exist in the database...  These
* records are the ones related to the
* entity, product and service tyeps.
*******************************************
*/
SELECT @zMessage = 'PSPriceWASPCalcSalesN, Deleting existing entries off WASPResolvedRouting...'
EXECUTE usp_Message @zMessage
DELETE
        FROM
                WASPResolvedRouting
        WHERE
                GasMonth=@GasMonthx AND
                NomOrActual=@WhichPricex AND
                EntityCID=@EntityCIDx AND
```

```
                        KProductID=@KProductIDx AND
                        KServiceID=@KServiceIDx
SELECT @zMessage = 'PSPriceWASPCalcSalesN, Finished deleting existing entries off WASPResolvedRouting...'
EXECUTE usp_Message @zMessage
/*
*******************************************
* Initially loop through the sales links
* found on the legdetail table (high level
* loop)... Only looping through those
* items that are associated with this
* entity and product/service type.
*******************************************
*/
SELECT @zPrevSalePKG=0
SELECT @zPrevSaleMID=0
EXECUTE usp_fLastDay @GasMonthx,@zLastDay OUTPUT
DECLARE LegDetailSaleCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
            SELECT
                        PurchasePKG,
                        RecMID,
                        DelMID,
                        SalesPKG,
                        Receipt,
                        LDIDPrev,
                        GasDay,
                        PurchasePointTID,
                        Step
                        FROM
                                    LegDetail
                        WHERE
                                    LegDetail.PurchasePointTID IN (SELECT DISTINCT TID FROM GasInv, Package,
K WHERE GasInv.PKG=Package.PKG AND k.kid = Package.KID AND GasInv.GasMonth=@GasMonthx AND
GasInv.DBCR=0 AND GasInv.PriceType=1 and Package.KProductID = @KProductIDx and Package.KServiceID =
@KServiceIDx AND K.EntityCID = @EntityCIDx) AND
                                    LegDetail.GasDay>=@GasMonthx AND
                                    LegDetail.GasDay<=@zLastDay AND
                                    LegDetail.GasMonth=@GasMonthx AND
                                    LegDetail.NomOrActuals=@WhichPricex AND
                                    LegDetail.LID=0 AND
                                    LegDetail.PurchasePKG>0 AND
                                    LegDetail.SalesPKG>0
                        ORDER BY
                                    LegDetail.SalesPKG,
                                    LegDetail.RecMID,
                                    LegDetail.PurchasePointTID,
                                    LegDetail.GasDay,
                                    LegDetail.PurchasePKG
SELECT @zMessage = 'PSPriceWASPCalcSalesN, opening main sales cursor (LegDetailSaleCursor)...'
EXECUTE usp_Message @zMessage
OPEN LegDetailSaleCursor
SELECT @zMessage = 'PSPriceWASPCalcSalesN, finished opening main sales cursor (LegDetailSaleCursor)...'
EXECUTE usp_Message @zMessage
FETCH NEXT FROM LegDetailSaleCursor INTO @yPurchasePKG,

            @yRecMID,@yDelMID,@ySalesPKG,@yReceipt,@yLDIDPrev,@yGasDay,@yPurchasePointTID,@yStep
WHILE @@FETCH_STATUS = 0
            BEGIN
                        /*
                        *******************************************
                        * Determine the classification of the
                        * purchase deal attached to this sales
                        * volume right here...
                        *******************************************
                        */
                        EXECUTE usp_fGetWaspIndicator @yPurchasePKG,@zIncludeInWasp OUTPUT
                        IF @zIncludeInWasp='Common'
                                    BEGIN
                                                SELECT @zDedicatedPurchasePKG=0
                                    END
                        ELSE
```

```
                    BEGIN
                              SELECT @zDedicatedPurchasePKG=@yPurchasePKG
                    END
          /*
          ******************************************
          * If sales package has changed OR
          * the meter within a sales package
          * has changed then (amongst other
          * things) sum up any/all other costs
          * for the meter (this ensures that only
          * one instance of other cost entries
          * are totaled for a given sales deal
          * at a given meter).
          ******************************************
          */
          SELECT @zOtherCostAmount=0
          IF (@ySalesPKG<>@zPrevSalePKG) OR (@yRecMID<>@zPrevSaleMID)
                    BEGIN
                              SELECT @zPrevSalePKG=@ySalesPKG
                              SELECT @zPrevSaleMID=@yRecMID
                              IF @WhichPricex=0
                                        BEGIN
                                                  SELECT
@zOtherCostAmount=ISNULL((SELECT SUM(Engine.Amount) FROM GasInv,Engine WHERE
GasInv.PKG=@ySalesPKG
                                                            AND
GasInv.GasMonth=@GasMonthx AND GasInv.PriceType=1 AND Engine.TID=GasInv.TID AND
GasInv.GasInv_MID=@yRecMID AND Engine.STID<>9),0)
                                        END
                              IF @WhichPricex=1
                                        BEGIN
                                                  SELECT
@zOtherCostAmount=ISNULL((SELECT SUM(Engine.AmountAct) FROM GasInv,Engine WHERE
GasInv.PKG=@ySalesPKG
                                                            AND
GasInv.GasMonth=@GasMonthx AND GasInv.PriceType=1 AND Engine.TID=GasInv.TID AND
GasInv.GasInv_MID=@yRecMID AND Engine.STID<>9),0)
                                        END
                    END
          /*
          ******************************************
          * Calculate the price and amount for the
          * sales item here (utilizing the Engine
          * calculation). The beginning volume is
          * the amount pulled off the sales association
          * on the database... Break from this
          * loop once the first price record has been
          * obtained (for this day)...
          ******************************************
          */
          SELECT @zPrice=0
          SELECT @zAmount=0
          SELECT @zVolume=0
          DECLARE EngineCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                    SELECT
                              Engine.PriceOrRateNom,
                              Engine.PriceOrRateAct
                              FROM
                                        GasInv,
                                        Engine
                              WHERE
                                        GasInv.PKG=@ySalesPKG AND
                                        GasInv.PriceType=1 AND
                                        Engine.TID=GasInv.TID AND
                                        GasInv.GasInv_MID=@yRecMID AND
                                        Engine.Effective<=@yGasDay AND
                                        Engine.STID=9
                              ORDER BY
                                        Engine.Effective DESC
          OPEN EngineCursor
```

```
        FETCH NEXT FROM EngineCursor INTO @xPriceOrRateNom,@xPriceOrRateAct
        IF @@FETCH_STATUS = 0
            BEGIN
                IF @zPrice=0
                    BEGIN
                    IF @WhichPricex=0
                            BEGIN
                                IF @GasMonthx <
'12/01/2000'
                                    BEGIN
                                    SELECT
@zPrice=ROUND(@xPriceOrRateNom,2)
                                    END
                                ELSE
                                    BEGIN
                                    SELECT
@zPrice=ROUND(@xPriceOrRateNom,4)
                                    END
                            END
                    ELSE
                            BEGIN
                                IF @GasMonthx <
'12/01/2000'
                                    BEGIN
                                    SELECT
@zPrice=ROUND(@xPriceOrRateAct,2)
                                    END
                                ELSE
                                    BEGIN
                                    SELECT
@zPrice=ROUND(@xPriceOrRateAct,4)
                                    END
                            END
                    SELECT @zVolume=@yReceipt
                    SELECT @zAmount=(@zVolume*@zPrice)
                    END
            END
        CLOSE EngineCursor
        DEALLOCATE EngineCursor
        /*
        ******************************************
        * Sum the other cost entry on the
        * amount brought back for the
        * production volume amount.  The
        * other cost entry will only have a
        * non zero value the first time a
        * sales meter is encountered. Make
        * sure to reset the price entry.
        ******************************************
        */
        IF @zOtherCostAmount<>0
            BEGIN
                SELECT @zAmount=@zAmount+@zOtherCostAmount
                IF (@zAmount<>0) AND (@zVolume<>0)
                    BEGIN
                        SELECT
@zPrice=ROUND((@zAmount/@zVolume),4)
                    END
            END
        /*
        ******************************************
        * Post a sales entry into the resolved
        * table here..  (LID=0)...  This will be
        * the starting point once the routing
        * interative process begins...
        ******************************************
        */
        IF ISNULL((SELECT count(*) FROM WASPResolvedRouting WHERE GasMonth=@GasMonthx
AND RecMID=@yRecMID AND DelMID=@yDelMID AND
```

```
                                                NomOrActual=@WhichPricex AND
IncludeInWasp=@zIncludeInWASP AND DedicatedPurchasePKG=@zDedicatedPurchasePKG AND
                                                ResolvedType='S' AND LID=0 AND
EntityCID=@EntityCIDx AND KProductID=@KProductIDx AND KServiceID=@KServiceIDx),0) < 1
                        BEGIN
                                INSERT
                                        INTO
                                                WASPResolvedRouting

        (GasMonth,RecMID,DelMID,NomOrActual,Receipt,FuelOrOther,Delivered,TransportAmount,GatheringAmou
nt,Amount,

        IncludeInWasp,DedicatedPurchasePKG,Price,ResolvedReceipt,ResolvedIndicator,ResolvedType,LID,Resolv
edDelivered,

        EntityCID,KProductID,KServiceID,ResolvedReceiptAmt,ResolvedDeliveredAmt)
                                        VALUES

        (@GasMonthx,@yRecMID,@yDelMID,@WhichPricex,@zVolume,0,@zVolume,0,0,@zAmount,

        @zIncludeInWASP,@zDedicatedPurchasePKG,@zPrice,0,'N','S',0,0,

        @EntityCIDx,@KProductIDx,@KServiceIDx,0,0)
                        END
                ELSE
                        BEGIN
                                IF (@zAmount<>0) AND (@zVolume<>0)
                                        BEGIN
                                                UPDATE
                                                        WASPResolvedRouting
                                                        SET

        Receipt=(Receipt+@zVolume),

        Delivered=(Delivered+@zVolume),

        Amount=(Amount+@zAmount),

        Price=ROUND(((Amount+@zAmount)/(Receipt+@zVolume)),4)
                                                        WHERE

        GasMonth=@GasMonthx AND
                                                                RecMID=@yRecMID
AND
                                                                DelMID=@yDelMID
AND

        NomOrActual=@WhichPricex AND

        IncludeInWasp=@zIncludeInWASP AND

        DedicatedPurchasePKG=@zDedicatedPurchasePKG AND
                                                                ResolvedType='S'
AND
                                                                LID=0 AND

        EntityCID=@EntityCIDx AND

        KProductID=@KProductIDx AND

        KServiceID=@KServiceIDx
                                        END
                        END
                FETCH NEXT FROM LegDetailSaleCursor INTO @yPurchasePKG,

        @yRecMID,@yDelMID,@ySalesPKG,@yReceipt,@yLDIDPrev,@yGasDay,@yPurchasePointTID,@yStep
        END
CLOSE LegDetailSaleCursor
DEALLOCATE LegDetailSaleCursor
/*
```

```
**********************************************
* Once all of the sales meters have been
* inserted then it is time to insert the
* transportation routing leg entries. THese
* are summarized entries.  No day-to-day
* cursor processing is required only the
* sum of the unique days.
*
* Transport legs (type 'T') and purchase
* points (type 'P') are posted here..
**********************************************
*/
DECLARE LegDetailChaseCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                LegDetail.PurchasePKG,
                LegDetail.LID,
                LegDetail.RecMID,
                LegDetail.DelMID,
                SUM(LegDetail.Receipt),
                SUM(LegDetail.Delivered),
                SUM(LegDetail.FuelOrOther),
                ROUND(SUM(LegDetail.Receipt*LegDetail.TransportationRate),2),
                ROUND(SUM(LegDetail.Receipt*LegDetail.GatheringRate),2)
                FROM
                        LegDetail
                WHERE
                        LegDetail.PurchasePointTID IN  (SELECT DISTINCT TID FROM GasInv, Package,
K WHERE GasInv.PKG=Package.PKG AND k.kid = Package.KID AND GasInv.GasMonth=@GasMonthx AND
GasInv.DBCR=0 AND GasInv.PriceType=1 and Package.KProductID = @KProductIDx and Package.KServiceID =
@KServiceIDx AND K.EntityCID = @EntityCIDx) AND
                        LegDetail.GasMonth=@GasMonthx AND
                        LegDetail.GasDay>=@GasMonthx AND
                        LegDetail.GasDay<=@zLastDay AND
                        LegDetail.NomOrActuals=@WhichPricex AND
                        LegDetail.SalesPKG=0
                GROUP BY
                        LegDetail.PurchasePKG,
                        LegDetail.LID,
                        LegDetail.RecMID,
                        LegDetail.DelMID
SELECT @zMessage =  'PSPriceWASPCalcSalesN, running query to create transportation legs...'
EXECUTE usp_Message @zMessage
SELECT @zPrevSalePKG=0
SELECT @zPrevSaleMID=0
SELECT @zMessage =  'PSPriceWASPCalcSalesN, opening cursor (LegDetailChaseCursor)...'
EXECUTE usp_Message @zMessage
OPEN LegDetailChaseCursor
SELECT @zMessage =  'PSPriceWASPCalcSalesN, finished opening cursor (LegDetailChaseCursor)...'
EXECUTE usp_Message @zMessage
FETCH NEXT FROM LegDetailChaseCursor INTO
@qPurchasePKG,@qLID,@qRecMID,@qDelMID,@qReceipt,@qDelivered,@qFuelOrOther,
                                                                        @qTransport,@qGathering
WHILE @@FETCH_STATUS = 0
        BEGIN
                /*
                **********************************************
                * Determine the classification of the
                * purchase deal attached to this transort
                * volume right here...
                **********************************************
                */
                IF (@qPurchasePKG<>@zPrevSalePKG) OR (@QLID<>@zPrevSaleMID)
                        BEGIN
                                SELECT @zPrevSalePKG=@qPurchasePKG
                                SELECT @zPrevSaleMID=@qLID
                        END
                EXECUTE usp_fGetWaspIndicator @qPurchasePKG,@zIncludeInWasp OUTPUT
                IF @zIncludeInWasp='Common'
                        BEGIN
                                SELECT @zDedicatedPurchasePKG=0
```

```
				END
			ELSE
				BEGIN
					SELECT @zDedicatedPurchasePKG=@qPurchasePKG
				END
			IF @qLID=0
				BEGIN
					SELECT @zType='P'
				END
			ELSE
				BEGIN
					SELECT @zType='T'
				END
			/*
			***********************************************
			* If the leg for this is in the WASP
			* temporary routing table then
			* just update the record with the
			* totals.  Otherwise, insert it and
			* go to the next leg...
			***********************************************
			*/
			SELECT @zAmountWithCosts=((@qGathering+@qTransport)*-1)
			IF ISNULL((SELECT count(*) FROM WASPResolvedRouting
					WHERE GasMonth=@GasMonthx AND RecMID=@qRecMID AND
	DelMID=@qDelMID AND
						NomOrActual=@WhichPricex AND
	IncludeInWasp=@zIncludeInWASP AND
						DedicatedPurchasePKG=@zDedicatedPurchasePKG AND
	ResolvedType=@zType AND LID=@qLID AND
						EntityCID=@EntityCIDx AND KProductID=@KProductIDx
	AND KServiceID=@KServiceIDx),0)<1
				BEGIN
					INSERT
						INTO
							WASPResolvedRouting

		(GasMonth,RecMID,DelMID,NomOrActual,Receipt,FuelOrOther,Delivered,

		TransportAmount,GatheringAmount,Amount,IncludeInWasp,DedicatedPurchasePKG,

		Price,ResolvedReceipt,ResolvedIndicator,ResolvedType,LID,ResolvedDelivered,

		EntityCID,KProductID,KServiceID,ResolvedReceiptAmt,ResolvedDeliveredAmt)
						VALUES

		(@GasMonthx,@qRecMID,@qDelMID,@WhichPricex,@qReceipt,@qFuelOrOther,@qDelivered,

		@qTransport,@qGathering,@zAmountWithCosts,@zIncludeInWASP,@zDedicatedPurchasePKG,
							0,0,'N',@zType,@qLID,0,

		@EntityCIDx,@KProductIDx,@KServiceIDx,0,0)
				END
			ELSE
				BEGIN
					UPDATE
						WASPResolvedRouting
						SET
							Receipt=(Receipt+@qReceipt),
							Delivered=(Delivered+@qDelivered),
							FuelOrOther=(FuelOrOther+@qFuelOrOther),

		TransportAmount=(TransportAmount+@qTransport),

		GatheringAmount=(GatheringAmount+@qGathering),
							Amount=(Amount+@zAmountWithCosts)
						WHERE
							GasMonth=@GasMonthx AND
							RecMID=@qRecMID AND
							DelMID=@qDelMID AND
```

```
                                                                NomOrActual=@WhichPricex AND
                                                                IncludeInWasp=@zIncludeInWASP AND
                DedicatedPurchasePKG=@zDedicatedPurchasePKG AND
                                                                ResolvedType=@zType AND
                                                                LID=@qLID AND
                                                                EntityCID=@EntityCIDx AND
                                                                KProductID=@KProductIDx AND
                                                                KServiceID=@KServiceIDx
                            END
                    FETCH NEXT FROM LegDetailChaseCursor INTO
@qPurchasePKG,@qLID,@qRecMID,@qDelMID,@qReceipt,@qDelivered,@qFuelOrOther,

                @qTransport,@qGathering
                END
CLOSE LegDetailChaseCursor
DEALLOCATE LegDetailChaseCursor
SELECT @zMessage = 'PSPriceWASPCalcSalesN Has Finished...'
EXECUTE usp_Message @zMessage
END









GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  ON    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceWASPClearMonth(
                                                                @GasMonthx DATETIME
                                                                )
AS
BEGIN
SET NOCOUNT ON
/*
*******************************************************************************************
Name: usp_PSPriceWaspClearMonth

Description: This routine will represents the common 'clean up' routine that
will purge anything on the database that can be purged.

The tables cleared include the following:

GasInvD (zero volume days for EstAct, Nom, PipelineActuals)
LegDetail (zero volume routing entries)

Inputs:

GasMonthx (gas month to calculate),

History:

06/30/1999 JAMIE   Original creation

08/04/1999 JAMIE   Modifications to not delete the entries in the
WASPPurchaseMeterTotals table.  This is becuase this table contains
the information necessary to calculate the margins on a deal.  All other
```

```
supporting table entries will be deleted.

10/12/1999 JAMIE Modifications to procedure to go out and delete any
daily gas inventory entries that contain no data. Again, since this procedure
is only executed when the gas month gets marked as completed there
should be no repurcussions except fewer database records to administer.
Anything of historical relevance will be retained (ie.. if any volume whatsoever).

03/30/2000 JAMIE Modifications made in the procedure to remove the zero entry
routing records from the database (prior deletion of the daily gas inventory
items should have deleted all of these (based on triggers). However,
this is for any/all other residuals.

08/25/2000 JAMIE Modified in order to remove obsolete cleanup tables
such as old routing tables/etc.

*****************************************************************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @zLastDay DATETIME

DECLARE @wTID INTEGER
DECLARE @wGasDay DATETIME

DECLARE @qLDID INTEGER

SELECT @zMessage = '**** STARTED, PSPriceWASPClearMonth'
EXECUTE usp_Message @zMessage
EXECUTE usp_fLastDay @GasMonthx,@zLastDay OUTPUT
/*
********************************************
* Remove daily inventory items that
* are now zero...
********************************************
*/
DECLARE GasInvDCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                GasInv.TID,
                GasInvD.GasDay
                FROM
                        GasInv,
                        GasInvD
                WHERE
                        GasInvD.TID = GasInv.TID AND
                        GasInv.GasMonth=@GasMonthx AND
                        GasInvD.EstAct = 0 AND
                        GasInvD.Nom = 0 AND
                        GasInvD.PipelineActuals = 0
                ORDER BY
                        GasInv.TID,
                        GasInvD.GasDay
SELECT @zMessage = ' PSPriceWASPClearMonth, Started removing ZEROd out Inventory Items...'
EXECUTE usp_Message @zMessage
OPEN GasInvDCursor
FETCH NEXT FROM GasInvDCursor INTO @wTID, @wGasDay
WHILE @@FETCH_STATUS = 0
        BEGIN
                BEGIN TRANSACTION
                DELETE FROM GasInvD WHERE TID=@wTID AND GasDay=@wGasDay
                COMMIT WORK
                FETCH NEXT FROM GasInvDCursor INTO @wTID, @wGasDay
        END
CLOSE GasInvDCursor
DEALLOCATE GasInvDCursor
SELECT @zMessage = ' PSPriceWASPClearMonth, Finished removing ZEROd out Inventory Items...'
EXECUTE usp_Message @zMessage
/*
********************************************
* Remove any routing items that had
* no entries within them.
```

```
******************************************
*/
DECLARE LegDetailCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
        SELECT
                LDID
                FROM
                        LegDetail
                WHERE
                        GasMonth=@GasMonthx AND
                        Receipt=0 AND
                        Delivered=0 AND
                        Balance=0 AND
                        FuelOrOther=0
                ORDER BY
                        PurchasePointTID
SELECT @zMessage = ' PSPriceWASPClearMonth, Started removing ZEROd out Routing (LegDetail) Items...'
EXECUTE usp_Message @zMessage
OPEN LegDetailCursor
FETCH NEXT FROM LegDetailCursor INTO @qLDID
WHILE @@FETCH_STATUS = 0
        BEGIN
                BEGIN TRANSACTION
                DELETE FROM LegDetail WHERE LDID=@qLDID
                COMMIT WORK
                FETCH NEXT FROM LegDetailCursor INTO @qLDID
        END
CLOSE LegDetailCursor
DEALLOCATE LegDetailCursor
SELECT @zMessage = ' PSPriceWASPClearMonth, Started removing ZEROd out Routing (LegDetail) Items...'
EXECUTE usp_Message @zMessage
SELECT @zMessage = '**** FINISHED, PSPriceWASPClearMonth'
EXECUTE usp_Message @zMessage
END



GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_PSPriceWASPDivieOutProceedsN(
                                                        @GasMonthx DATETIME,
                                                        @WhichPricex INTEGER,
                                                        @EntityCIDx VARCHAR(12)
                                                                )
AS
BEGIN
/*
******************************************************************************************
Name: usp_PSPriceWASPDivieOutProceeds

Description:

This procedure will get executed during the WASP calculation in order
to credit the financial proceeds (gain or loss) from one deal to another.

These proceed designations are setup on the package table
(FinancialPKG and FinancialMID field contains either a deal id
or a common wasp meter pool point that is to receive the proceeds).
These fields are mutually exclusive on the deal table.

The default for all deals is the deal itself (for owning the proceeds). Only
if the FinancialPKG or FinancialMID field has been entered will it be
distributed elsewhere.  The distribution amount (if any) will be posted
on the from deal record (either in the FinancialNomAmount or
FinancialActAmount field, dependant on which price is calculating).
```

```
This procedure works for 3rd party deals only (deal classification rule
is equal to 'None').   The reason for this is because these are the only
types of deals where we know the actual margin ('Common' (Wasp)
and sanctioned sales (Dedicated) are netback calculated deals.

For all FinancialPKG/MID entries this procedure will:

1. Calculate the margin (purchase price and purchase meter price).
2. Reduce the purchase meter amounts by the amount calculated.
3. Post the dollar amount to the proceed purchase meter(s) based on their respective
volume weightings to the deal.

Inputs:

GasMonthx - Gas Month
WhichPricex - 0=Nominations, 1=Actuals
EntityCIDx - owning company/entity

History:

07/27/1999 JAMIE Original Creation.

10/13/1999 JAMIE Modified to cast the distribution amounts to decimal(18,4).
This is because of bug receiving correct amount to distrubute when dividing
two integers.

03/30/2000 JAMIE Modified the program to not use the 'PackageLinks'
table but to use the FinancialPKG field stored on the deal table. This
was done as part of the integration with linking and the new route
process.

05/24/2000 JAMIE Modified to include the owning company/entity.

07/28/2000 JAMIE Modified in order to post the updates of what is
being distributed back to the Package table (for the 'from' deal) and
then post the amounts to the WASP Purchase Meter table (for deals)
or WASP Legs for meters.  This change was done in order to
facilitate the reordering of the calculations.

08/07/2000 JAMIE Modified so that even if diving to a specific deal IF
that deal is a wasp deal then all deals that share the same original
purchase point meters as the deal being divied to (in the 'Common' pool)
will share in the divie.

08/18/2000 JAMIE Modified so that if diving to a specific deal then the
amount will go to the WASPResolvedRouting table versus the obsolete
WASPPurchaseMeterTable.

*****************************************************************************************
*/
/*
*******************************************
* Declare all variables and cursors
* that are needed by this process.
*******************************************
*/
DECLARE @zMessage VARCHAR(254)
DECLARE @zLastDay DATETIME
DECLARE @zPurchasePrice DECIMAL(19,6)
DECLARE @zIncludeInWasp VARCHAR(10)
DECLARE @zTotalVolume INTEGER
DECLARE @zGrandTotalDistributed DECIMAL(19,2)
DECLARE @zTempVolPercent DECIMAL(19,4)
DECLARE @zAmountToDistribute DECIMAL(19,2)
DECLARE @zMarginPrice DECIMAL(18,4)
DECLARE @zMarginAmt DECIMAL(19,2)
DECLARE @zFoundDedicated VARCHAR(1)
DECLARE @zSumofFBOPKGCreditMeters INTEGER
DECLARE @zAmountToCredit DECIMAL(19,2)
DECLARE @zSumofFBOPKGMeters INTEGER

DECLARE @yPKG INTEGER
DECLARE @yFinancialPKG INTEGER
DECLARE @yKProductID INTEGER
DECLARE @yKServiceID INTEGER
DECLARE @yFinancialMID INTEGER

DECLARE @yWASPReceipt DECIMAL(19,2)
DECLARE @yWASPAmount DECIMAL(19,2)
DECLARE @yWASPPrice DECIMAL(19,6)
DECLARE @yWASPResolvedID INTEGER

DECLARE @yWASPCreditReceipt DECIMAL(19,2)
DECLARE @yWASPCreditAmount DECIMAL(19,2)
DECLARE @yWASPCreditPrice DECIMAL(19,2)
DECLARE @yWASPCreditResolvedID INTEGER

DECLARE @qDelivered DECIMAL(19,2)
DECLARE @qAmount DECIMAL(19,2)
DECLARE @qPrice DECIMAL(19,6)
DECLARE @qResolvedID INTEGER

SELECT @zMessage = 'PSPriceWASPDivieOutProceedsN, ***STARTED***'
EXECUTE usp_Message @zMessage
EXECUTE usp_fLastDay @GasMonthx,@zLastDay OUTPUT
/*
*******************************************
* At this point we want to loop
* through all of the packages
* (deals) on the system that had
* requested that the proceeds
* be divied to other deals.
*******************************************
*/

DECLARE ProceedsCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
          SELECT
                    PKG,
                    FinancialPKG,
                    KProductID,
                    KServiceID,
                    FinancialMID
                    FROM
                              Package,
                              K
                    WHERE
                              (K.KID=Package.KID) AND
                              (K.EntityCID=@EntityCIDx) AND
                              (StartDate BETWEEN @GasMonthx AND @zLastDay) AND
                              (((FinancialPKG IS NOT NULL) AND (FinancialPKG<>0)) OR ((FinancialMID IS
NOT NULL) AND (FinancialMID<>0)))
                    ORDER BY
                              PKG

OPEN ProceedsCursor
FETCH NEXT FROM ProceedsCursor INTO @yPKG,@yFinancialPKG,@yKProductID,@yKServiceID,@yFinancialMID
WHILE @@FETCH_STATUS = 0
          BEGIN
                    BEGIN TRANSACTION
                    SELECT @zMessage = 'PSPriceWASPDivieOutProceedsN,  Proceeds divied from deal...' +
CAST(@yPKG as VARCHAR(12))
                    EXECUTE usp_Message @zMessage
                    /*
                    *******************************************
                    * Get the agreed upon purchase
                    * price from the engine for the
                    * 'from' purchase deal. The total
                    * volume across all days is also
                    * obtained here (for all meters).
```

```
                        *
                        * Base the price on the weighted
                        * averages for all entries within
                        * the Engine table.
                        *
                        * This yields the single weighted
                        * average cost across all wells
                        * and days.
                        *
                        * This price should be the price
                        * that was found PRIOR to diving
                        * out any adjusments.
                        ******************************************
                        */
                        IF @WhichPricex=0
                                BEGIN
                                        SELECT @zPurchasePrice=ROUND(ISNULL((SELECT
SUM(Engine.Amount)/SUM(Engine.Volume) FROM Engine,GasInv
                                                                                WHERE
(GasInv.GasMonth=@GasMonthx AND GasInv.PKG=@yPKG) AND (Engine.TID=GasInv.TID) AND (Engine.STID=8)
AND

        Engine.Amount>0 and Engine.Volume>0),0),4)
                                        SELECT @zTotalVolume=ISNULL((SELECT SUM(Engine.Volume)
FROM Engine,GasInv WHERE (GasInv.GasMonth=@GasMonthx AND GasInv.PKG=@yPKG) AND

        (Engine.TID=GasInv.TID) AND (Engine.STID=8) AND

        Engine.Amount>0 and Engine.Volume>0),0)
                                END
                        IF @WhichPricex=1
                                BEGIN
                                        SELECT @zPurchasePrice=ROUND(ISNULL((SELECT
SUM(Engine.AmountAct)/SUM(Engine.VolumeAct) FROM Engine,GasInv
                                                                                WHERE
(GasInv.GasMonth=@GasMonthx AND GasInv.PKG=@yPKG) AND (Engine.TID=GasInv.TID) AND (Engine.STID=8)
AND

        Engine.AmountAct>0 and Engine.VolumeAct>0),0),4)
                                        SELECT @zTotalVolume=ISNULL((SELECT SUM(Engine.VolumeAct)
FROM Engine,GasInv WHERE (GasInv.GasMonth=@GasMonthx AND GasInv.PKG=@yPKG) AND

        (Engine.TID=GasInv.TID) AND (Engine.STID=8) AND

        Engine.AmountAct>0 and Engine.VolumeAct>0),0)
                                END
                        /*
                        ******************************************
                        * Only continue if the purchase
                        * price (average) for this deal
                        * could be calculated (ie.. there
                        * was a volume and there was
                        * a price entry.
                        *
                        * Now loop through each of the
                        * meters to determine how much
                        * to reduce each meter by...
                        ******************************************
                        */
                        SELECT @zGrandTotalDistributed=0
                        IF (@zPurchasePrice>0)
                                BEGIN
                                        IF @zTotalVolume<>0
                                                BEGIN

                                                        /* This cursor is for determining proceed
amounts*/

                                                        DECLARE
WASPResolvedRoutingDebitCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
```

```
                                                            SELECT
                                                                  receipt,
                                                                  amount,
                                                                  price,
                                                                  ResolvedID
                                                                  FROM
      WASPResolvedRouting
                                                                  WHERE
      GasMonth=@GasMonthx AND
      DedicatedPurchasePKG=@yPKG AND
      NomOrActual=@WhichPricex AND
      EntityCID=@EntityCIDx AND
      KProductID=@yKProductID AND
      KServiceID=@yKServiceID AND
      ResolvedType='P' AND
                                                                        LID=0
AND
      RecMID=DelMID
                                                      OPEN WASPResolvedRoutingDebitCursor
                                               FETCH NEXT FROM WASPResolvedRoutingDebitCursor
INTO @yWASPReceipt,@yWASPAmount,
      @yWASPPrice,@yWASPResolvedID
                                                      WHILE @@FETCH_STATUS = 0
                                                            BEGIN
                                                            SELECT
@zMarginPrice=ROUND((@yWASPPrice-@zPurchasePrice),4)
                                                            SELECT
@zMarginAmt=ROUND((@zMarginPrice*@zTotalVolume),2)
                                                            IF @yWaspReceipt>0
                                                                  BEGIN
      SELECT @zTempVolPercent=ROUND((@yWaspReceipt/@zTotalVolume),4)
      SELECT @zAmountToDistribute=ROUND((@zTempVolPercent*@zMarginAmt),2)
      SELECT @zGrandTotalDistributed=@zGrandTotalDistributed+@zAmountToDistribute
      UPDATE
            WASPResolvedRouting
            SET
                  Amount=Amount+(@zAmountToDistribute*-1)
            WHERE
                  ResolvedID=@yWASPResolvedID
      UPDATE
            WASPResolvedRouting
            SET
                  Price=(Amount/Receipt)
            WHERE
                  ResolvedID=@yWASPResolvedID AND
```

```
                        Receipt<>0 AND

                        Amount<>0
                                                        END
                                                    FETCH NEXT FROM
WASPResolvedRoutingDebitCursor INTO @yWASPReceipt,@yWASPAmount,

                        @yWASPPrice,@yWASPResolvedID
                                                    END
                                                CLOSE WASPResolvedRoutingDebitCursor
                                                DEALLOCATE
WASPResolvedRoutingDebitCursor
                                        END
                        END
                    /*
                    ******************************************
                    * At this point, if there has been any
                    * proceeds distributed from the
                    * purchase deal then go and distribute
                    * the amount back to the deal where
                    * that is receiving credit.  This is
                    * based on the volume weighting
                    * distribution at the target 'to' meter.
                    *
                    * The field zGrandTotalDistributed contains
                    * the total dollar amount to be credited
                    * the the meters (based on volume
                    * weighting.
                    ******************************************
                    */
                    IF @zGrandTotalDistributed<>0
                        BEGIN
                            /*
                            ******************************************
                            * Post the 'from' deal with the
                            * appropriate distributed amount.
                            * This is the total amount across
                            * the entire deal and is stored on
                            * the deal record to provide an
                            * audit of how much was diverted.
                            ******************************************
                            */
                            IF @WhichPricex=0
                                BEGIN
                                    UPDATE
                                        Package
                                        SET
    FinancialNomAmount=@zGrandTotalDistributed
                                        WHERE
                                            PKG=@yPKG
                                END
                            IF @WhichPricex=1
                                BEGIN
                                    UPDATE
                                        Package
                                        SET
    FinancialActAmount=@zGrandTotalDistributed
                                        WHERE
                                            PKG=@yPKG
                                END
                            /*
                            ******************************************
                            * If diving to another deal then
                            * perform this....  Adjustments are
                            * made to the WASPResolvedRouting
                            * table. There is no need to post
                            * adjustments to the Engine table
```

```
                                   * since the target deals have either
                                   * not yet calculated (dedicated) or
                                   * the Engine price is fixed (3rd
                                   * party).
                                   ********************************************
                                   */
                                   IF (@yFinancialPKG IS NOT NULL) AND (@yFinancialPKG<>0)
                                        BEGIN
                                        /*
                                        ********************************************
                                        * Determine if the target deal is
                                        * a wasp deal or a 3rd party or
                                        * sanctioned sale deal... If it is a
                                        * wasp deal then the originating
                                        * meters in the common pool
                                        * will get the credit.
                                        ********************************************
                                        */
                                        EXECUTE usp_fGetWaspIndicator
@yFinancialPKG,@zIncludeInWasp OUTPUT
                                        /*
                                        ********************************************
                                        * Sum totals across all meters on
                                        * the target deal...
                                        ********************************************
                                        */
                                        IF @WhichPricex=0
                                                  BEGIN
                                                  SELECT
@zSumofFBOPKGCreditMeters=ISNULL((SELECT SUM(inventory.Nom) FROM gasInv AS inventory

          WHERE inventory.PKG=@yFinancialPKG AND inventory.GasMonth=@GasMonthx AND
inventory.DBCR=0 AND inventory.PriceType=1),0)
                                                  END
                                        IF @WhichPricex=1
                                                  BEGIN
                                                  SELECT
@zSumofFBOPKGCreditMeters=ISNULL((SELECT SUM(inventory.PipelineActuals) FROM gasInv AS inventory

          WHERE inventory.PKG=@yFinancialPKG AND inventory.GasMonth=@GasMonthx AND
inventory.DBCR=0 AND inventory.PriceType=1),0)
                                                  END
                                        /*
                                        ********************************************
                                        * If there is some sort of volume
                                        * then post it proportionately.
                                        ********************************************
                                        */
                                        IF @zSumofFBOPKGCreditMeters<>0
                                             BEGIN
                                                  /*

     ********************************************

                                                  * if not a wasp deal to post the
                                                  * credit to then...

     ********************************************

                                                  */
                                                  IF @zIncludeInWasp <>
'Common'
                                                            BEGIN


     /* This cursor is for posting proceeds to a dedicated deal point*/


     DECLARE WASPResolvedRoutingCreditDedicatedCursor CURSOR LOCAL STATIC FORWARD_ONLY
FOR

          SELECT
```

```
                    receipt,
                    amount,
                    price,
                    ResolvedID
                    FROM
                            WASPResolvedRouting
                    WHERE
                            GasMonth=@GasMonthx AND
                            DedicatedPurchasePKG=@yFinancialPKG AND
                            NomOrActual=@WhichPricex AND
                            receipt>0 AND
                            ResolvedType='P' AND
                            LID=0 AND
                    RecMID=DelMID
        OPEN WASPResolvedRoutingCreditDedicatedCursor
        FETCH NEXT FROM WASPResolvedRoutingCreditDedicatedCursor INTO @yWASPCreditReceipt,
                @yWASPCreditAmount,@yWASPCreditPrice,@yWASPCreditResolvedID
        WHILE @@FETCH_STATUS = 0
                BEGIN
                    SELECT
@zTempVolPercent=ROUND((@yWaspCreditReceipt/@zSumofFBOPKGCreditMeters),4)
                    SELECT
@zAmountToCredit=ROUND((@zTempVolPercent*@zGrandTotalDistributed),2)
                    IF @zAmountToCredit<>0
                    BEGIN
                            UPDATE
                                            WASPResolvedRouting
                                            SET
                                            Amount=(Amount+@zAmountToCredit)
                                    WHERE
        ResolvedID=@yWASPCreditResolvedID
                            UPDATE
                                            WASPResolvedRouting
                                            SET
                                                    Price=(Amount/Receipt)
```

```
                                        WHERE
                ResolvedID=@yWASPCreditResolvedID AND
                                                Amount<>0 AND
                                                Receipt<>0
                        END
                        FETCH NEXT FROM WASPResolvedRoutingCreditDedicatedCursor INTO
@yWASPCreditReceipt,
                @yWASPCreditAmount,@yWASPCreditPrice,@yWASPCreditResolvedID
                    END
                CLOSE WASPResolvedRoutingCreditDedicatedCursor
                DEALLOCATE WASPResolvedRoutingCreditDedicatedCursor
                                                            END
                                                /*
                *******************************************
                                                * if wasp deal to post the
                                                * credit to then...
                *******************************************
                                                */
                                                IF @zIncludeInWasp='Common'
                                                            BEGIN
                /* This cursor is for posting proceeds to a common meter purchase point*/
                DECLARE WASPResolvedRoutingCreditWASPCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                    SELECT
                        wp.receipt,
                        wp.amount,
                        wp.price,
                        wp.ResolvedID
                        FROM
                            WASPResolvedRouting AS wp,
                            GasInv AS g
                        WHERE
                            g.GasMonth=@GasMonthx AND
                            g.PKG=@yFinancialPKG AND
                            g.GasInv_MID=wp.RecMID AND
                            wp.GasMonth=@GasMonthx AND
                            wp.DedicatedPurchasePKG=0 AND
                            wp.IncludeInWasp='Common' AND
```

```
                                        wp.NomOrActual=@WhichPricex AND
                                        wp.receipt>0 AND
                                        wp.EntityCID=@EntityCIDx AND
                                        wp.KProductID=@yKProductID AND
                                        wp.KServiceID=@yKServiceID AND
                                        wp.ResolvedType='P' AND
                                        wp.LID=0 AND
                                        wp.RecMID=DelMID
            OPEN WASPResolvedRoutingCreditWASPCursor
            FETCH NEXT FROM WASPResolvedRoutingCreditWASPCursor INTO @yWASPCreditReceipt,
                        @yWASPCreditAmount,@yWASPCreditPrice,@yWASPCreditResolvedID
            WHILE @@FETCH_STATUS = 0
                  BEGIN
                        SELECT
      @zTempVolPercent=ROUND((@yWaspCreditReceipt/@zSumofFBOPKGCreditMeters),4)
                        SELECT
      @zAmountToCredit=ROUND((@zTempVolPercent*@zGrandTotalDistributed),2)
                        IF @zAmountToCredit<>0
                        BEGIN
                              UPDATE
                                          WASPResolvedRouting
                                          SET
                                          Amount=(Amount+@zAmountToCredit)
                                    WHERE
            ResolvedID=@yWASPCreditResolvedID
                              UPDATE
                                          WASPResolvedRouting
                                          SET
                                                Price=(Amount/Receipt)
                                    WHERE
            ResolvedID=@yWASPCreditResolvedID AND
                                                Amount<>0 AND
                                                Receipt<>0
                        END
```

```
                                        FETCH NEXT FROM WASPResolvedRoutingCreditWASPCursor INTO
@yWASPCreditReceipt,

          @yWASPCreditAmount,@yWASPCreditPrice,@yWASPCreditResolvedID

                    END

          CLOSE WASPResolvedRoutingCreditWASPCursor

          DEALLOCATE WASPResolvedRoutingCreditWASPCursor
                                                                                        END
                                                                  END
                                                        END
                                              /*
                                              ******************************************
                                              * If diving to the WASP pool then
                                              * the total distributed is posted
                                              * proportionately on each leg that
                                              * contains this meter in the
                                              * 'Common' pool.
                                              ******************************************
                                              */
                                              IF (@yFinancialMID IS NOT NULL) AND (@yFinancialMID<>0)
                                                        BEGIN
                                                        /*
                                                        ******************************************
                                                        * Sum totals across all legs that
                                                        * have the same meter in the
                                                        * 'Common' pool for the month.
                                                        ******************************************
                                                        */
                                                        SELECT
@zSumofFBOPKGCreditMeters=ISNULL((SELECT SUM(Delivered) FROM WaspResolvedRouting

                    WHERE GasMonth=@GasMonthx AND LID<>0 AND

                              NomOrActual=@WhichPricex AND IncludeInWasp='Common' AND

                              EntityCID=@EntityCIDx AND KProductID=@yKProductID AND

                              KServiceID=@yKServiceID AND DelMID=@yFinancialMID),0)
                                                        /*
                                                        ******************************************
                                                        * If there is some sort of volume
                                                        * then post it proportionately to
                                                        * each of the legs in the WASP
                                                        * resolved routing table.
                                                        ******************************************
                                                        */
                                                        IF @zSumofFBOPKGCreditMeters<>0
                                                                  BEGIN

                                                                                   /* This cursor is for
posting proceeds to a wasp pool (non entry point)*/

                                                                                   DECLARE
WASPResolvedRoutingCreditCursor CURSOR LOCAL STATIC FORWARD_ONLY FOR
                                                                                             SELECT

          delivered,

          amount,

          price,

          ResolvedID

          FROM
```

```
                    WASPResolvedRouting
          WHERE
                    GasMonth=@GasMonthx AND
                    NomOrActual=@WhichPricex AND
                    EntityCID=@EntityCIDx AND
                    KProductID=@yKProductID AND
                    KServiceID=@yKServiceID AND
                    DelMID=@yFinancialMID AND
                    LID<>0 AND
                    IncludeInWasp='Common' AND
                    delivered>0
                                                                      OPEN
WASPResolvedRoutingCreditCursor
                                                                              FETCH NEXT FROM
WASPResolvedRoutingCreditCursor INTO @qDelivered,@qAmount,@qPrice,@qResolvedID
                                                                              WHILE
@@FETCH_STATUS = 0
                                                                                      BEGIN
          /*
          ********************************************
          * Determine the percent to post
          * here...
          ********************************************
          */
          SELECT @zTempVolPercent=ROUND((@qDelivered/@zSumofFBOPKGCreditMeters),4)
          SELECT @zAmountToCredit=ROUND((@zTempVolPercent*@zGrandTotalDistributed),2)
          IF @zAmountToCredit<>0
                    BEGIN
                              UPDATE
                                        WASPResolvedRouting
                                        SET
                                                  Amount=(Amount+@zAmountToCredit)
                                        WHERE
                                                  ResolvedID=@qResolvedID
                    END
          FETCH NEXT FROM WASPResolvedRoutingCreditCursor INTO @qDelivered,@qAmount,
          @qPrice,@qResolvedID
                                                                                      END
```

```
                                                                                CLOSE
WASPResolvedRoutingCreditCursor
                                                                                DEALLOCATE
WASPResolvedRoutingCreditCursor
                                                            END
                                                    END
                                END
                        COMMIT WORK
                        FETCH NEXT FROM ProceedsCursor INTO
@yPKG,@yFinancialPKG,@yKProductID,@yKServiceID,@yFinancialMID
                END
CLOSE ProceedsCursor
DEALLOCATE ProceedsCursor
SELECT @zMessage = 'PSPriceWASPDivieOutProceedsN, ***FINISHED***'
EXECUTE usp_Message @zMessage
END





GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  ON    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE usp_fGetCalcIndex(
                                        @TIDx INTEGER,
                                        @NomOrActualx INTEGER,
                                        @EntityCIDx VARCHAR(12),
                                        @KProductIDx INTEGER,
                                        @KServiceIDx INTEGER,
                                        @GasMonthx DATETIME,
                                        @rReturnValue DECIMAL(19,6) OUTPUT
                                                )
AS
BEGIN
/*
*******************************************************************************************
Name: usp_fGetCalcIndex

Description: This is the main process for finding the actual price that was
calculated for a WASP purchase deal.  The WASPResolvedRouting table
contains all of the prices for WASP purchases.

An attempt should first be made to see if the price can be resolved by reading
for a 'Dedicated' wasp pool (sanctioned sales/purchases are more or less
dedicated).  The purchase deal id must match the dedicatedpurchasepkg field
on the WASPResolvedRouting.

If the specific package cannot be found then the purchase meter will be used
(ie..'Common' wasp pool).

Inputs:

TIDx - Unique Key to gas inventory record (GasInv)
NomOrActualx -  0=Nominations, 1=Actualizations
EntityCIDx - owner
KProductIDx - product id
KServiceIDx - service
GasMonthx - Current gas month
rReturnValue - OUTPUT return value
```

```
History:

06/29/1999 JAMIE   Modified from original creation
(date of original creation ?) to  support WASP calc changes V2.20.

06/22/2000 JAMIE Modified to get wasp prices based on entity,
product, and service.

08/18/2000 JAMIE Modified to get the wasp prices off the WASPResolvedRouting
table versus the obsolete WASPPurchaseMeterTable.

11/07/2000 JAMIE Modifications to convert from Watcom-SQL to
Transact-SQL.

*******************************************************************************************
*/
/*
********************************************
* Declare all variables and cursors
* that are needed by this process.
********************************************
*/
DECLARE @ymid INTEGER
DECLARE @ypkg INTEGER
DECLARE @ygasmonth DATETIME
DECLARE @yWorkValue DECIMAL(19,6)
DECLARE @message VARCHAR(255)
/*
********************************************
* Initialize key fields and then get
* the meter and deal information
* off the gas inventory table.
********************************************
*/
SELECT @rReturnValue=0
SELECT
          @ymid=gasinv_mid,
          @ypkg=pkg,
          @ygasmonth=gasmonth
          FROM
                    gasinv
          WHERE
                    tid=@tidx
/*
********************************************
* Now try and read the price off the
* WASPResolvedRouting with
* an assumption that it could be a
* sanctioned sale deal.
********************************************
*/
/*
********************************************
* If price is a dedicated purchase
* price then get that number. Otherwise,
* the the price from the WASP pool.
********************************************
*/
IF ((SELECT count(*) FROM WASPResolvedRouting
          WHERE DedicatedPurchasePKG=@ypkg AND GasMonth=@ygasmonth AND IncludeInWasp='Dedicated'
AND NomOrActual=@NomOrActualx AND RecMID=@ymid
                              AND DelMID=@ymid AND ResolvedType='P' AND LID=0 AND
EntityCID=@EntityCIDx AND KProductID=@KProductIDx AND KServiceID=@KServiceIDx) > 0)
          BEGIN
                    SELECT @yWorkValue=Price FROM WASPResolvedRouting
                              WHERE DedicatedPurchasePKG=@ypkg AND GasMonth=@ygasmonth AND
IncludeInWasp='Dedicated' AND NomOrActual=@NomOrActualx AND RecMID=@ymid
                                        AND DelMID=@ymid AND ResolvedType='P' AND LID=0 AND
EntityCID=@EntityCIDx AND KProductID=@KProductIDx AND KServiceID=@KServiceIDx
```

```
          END
ELSE
          BEGIN
                    SELECT @yWorkValue=Price FROM WASPResolvedRouting
                              WHERE RecMID=@ymid AND DelMID=@ymid AND LID=0 AND ResolvedType='P'
                                        AND gasmonth=@ygasmonth AND IncludeInWasp='Common' AND
NomOrActual=@NomOrActualx AND EntityCID=@EntityCIDx
                                                  AND KProductID=@KProductIDx AND
KServiceID=@KServiceIDx
          END
/*
*******************************************
* If some sort of price was found then
* return with it... Otherwise zeros
* are returned (no price calculated).
*******************************************
*/
/*
SELECT @message = 'WASP Price ' +
                              CAST(@yWorkValue AS VARCHAR(12)) +
                              ' for meter id ' +
                              CAST(@ymid AS VARCHAR(12))
EXECUTE usp_message @message
*/
IF @yWorkValue IS NOT NULL
          BEGIN
                    SELECT @rReturnValue=@yWorkValue
          END
END


GO
SET QUOTED_IDENTIFIER  OFF   SET ANSI_NULLS  ON
GO
```

ADDITIONAL FEATURES

The present invention has been disclosed, illustrated, and described in relation to a client-server application that facilitates pricing and distribution of fuel to a customer. Although centralized data storage and manipulation is preferred in regard to the version of the system that has been provided, the inventors contemplate other applications and enhancements that certainly are within the scope of the present invention. For example, the present invention relies on data inputs and feeds from a variety of entities such as producers, transporters, etc. Although such data inputs are often entered manually into the systems provided by the present invention, such data inputs could be automatically delivered and stored within data store 106 (FIG. 2). For example, transporters controlling actual meters along a gas pipeline, for example, could be outfitted with remote sensors and transmitters that provide shipment volume, etc. details directly to the systems provided by the present invention. Moreover, data inputs such as indexing datum used to drive pricing, etc. may be similarly obtained. And, since such data inputs can come from a variety of sources, modern communications technologies such as the Internet, wireless technologies, etc. could all be used to couple an operator of the systems and methods provided by the present invention with such sources. Accordingly, the present invention is not limited to any particular data retrieval system, topology, method, or paradigm. Those skilled in the art will be immediately able to adapt and modify the underlying data collection capabilities of the systems and methods provided by the present invention to incorporate such new and modern technologies and techniques.

Finally, it should be noted that the present invention contemplates and provides for an elaborate reporting capability as provided within the software contained on the attached compact disc. Those skilled in the art of computer programming and those familiar with fuel deal management will immediately understand that any number of report may be prepared to suit and satisfy management requirements. The database tables maintained by the present invention certainly support all types of relational type queries that such reports may require.

Thus, having fully described the present invention by way of example with reference to the attached drawing figures, it will be readily appreciated that many changes and modifications may be made to the invention and to any of the exemplary embodiments shown and/or described herein without departing from the spirit or scope of the invention which is defined in the appended claims.